UNIVERSIDADE DE LISBOA

FACULDADE DE CIÊNCIAS

Bus Driver Rostering by Hybrid Methods Based on Column Generation

Doutoramento em Informática

Especialidade de Engenharia Informática

Victor Manuel Meneses Barbosa

Tese orientada por:

Ana Luísa do Carmo Correia Respício

Filipe Pereira Pinto da Cunha e Alvelos

Documento especialmente elaborado para a obtenção do grau de doutor

2018

UNIVERSIDADE DE LISBOA

FACULDADE DE CIÊNCIAS

Bus Driver Rostering by Hybrid Methods Based on Column Generation

Doutoramento em Informática

Especialidade de Engenharia Informática

Victor Manuel Meneses Barbosa

Tese orientada por:

Ana Luísa do Carmo Correia Respício

Filipe Pereira Pinto da Cunha e Alvelos

Júri:

Presidente:

- Doutor Nuno Fuentecilla Maia Ferreira Neves, Professor Catedrático,
  Faculdade de Ciências da Universidade de Lisboa

Vogais:

- Doutor José Fernando da Costa Oliveira, Professor Catedrático
  Faculdade de Engenharia da Universidade do Porto
- Doutor Amaro Fernandes de Sousa, Professor Auxiliar
  Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro
- Doutora Margarida Maria Gonçalves Vaz Pato, Professora Catedrática
  Instituto Superior de Economia e Gestão da Universidade de Lisboa
- Doutora Ana Luísa do Carmo Correia Respício, Professora Auxiliar
  Faculdade de Ciências da Universidade de Lisboa
- Doutor André Osório e Cruz de Azeredo Falcão, Professor Auxiliar
  Faculdade de Ciências da Universidade de Lisboa

Documento especialmente elaborado para a obtenção do grau de doutor

2018

To Rodrigo and Mara

Abstract

Rostering problems arise in a diversity of areas where, according to the business and labor rules, distinct variants of the problem are obtained with different constraints and objectives considered. The diversity of existing rostering problems, allied with their complexity, justifies the activity of the research community addressing them. The current research on rostering problems is mainly devoted to achieving near-optimal solutions since, most of the times, the time needed to obtain optimal solutions is very high.

In this thesis, a Bus Driver Rostering Problem is addressed, to which an integer programming model is adapted from the literature, and a new decomposition model with three distinct subproblems representations is proposed. The main objective of this research is to develop and evaluate a new approach to obtain solutions to the problem in study. The new approach follows the concept of search based on column generation, which consists in using the column generation method to solve problems represented by decomposition models and, after, applying metaheuristics to search for the best combination of subproblem solutions that, when combined, result in a feasible integer solution to the complete problem.

Besides the new decomposition models proposed for the Bus Driver Rostering Problem, this thesis proposes the extension of the concept of search by column generation to allow using population-based metaheuristics and presents the implementation of the first metaheuristic using populations, based on the extension, which is an evolutionary algorithm.

There are two additional contributions of this thesis. The first is an heuristic allowing to obtain solutions for the subproblems in an individual or aggregated way and the second is a repair operator which can be used by the metaheuristics to repair infeasible solutions and, eventually, generate missing subproblem solutions needed.

The thesis includes the description and results from an extensive set of computational tests. Multiple configurations of the column generation with three decomposition models are tested to assess the best configuration to use in the generation of the search space for the metaheuristic. Additional tests compare distinct single-solution metaheuristics and our basic evolutionary

algorithm in the search for integer solutions in the search space obtained by the column generation. A final set of tests compares the results of our final algorithm (with the best column generation configuration and the evolutionary algorithm using the repair operator) and the solutions obtained by solving the problem represented by the integer programming model with a commercial solver.

Keywords: Rostering; Metaheuristics; Evolutionary Algorithms; Column Generation;

Resumo

Os problemas de definição de escalas de pessoal (*Rostering*) surgem em diversas áreas de negócio onde, de acordo com regras de cada negócio em particular e a legislação laboral aplicável, podem surgir variantes do problema que consideram, na modelação do problema, restrições e objetivos distintos. A diversidade de problemas de *Rostering* existentes, aliada à sua complexidade, justifica a atividade da comunidade na investigação destes problemas. Atualmente, grande parte da investigação com problemas de *Rostering* está direcionada para a obtenção de soluções aproximadas (da solução ótima) uma vez que, a maior parte das vezes, o tempo necessário para obter a solução ótima é muito elevado.

A investigação descrita nesta tese pretende dar resposta à seguinte questão: "Será que um algoritmo híbrido que combina o método de geração de colunas com meta-heurísticas é eficaz na obtenção de soluções de qualidade para problemas de *Rostering* num intervalo de tempo razoável?"

A abordagem proposta segue o conceito de "*Search by Column Generation*" e o problema que foi utilizado na investigação é um problema de definição de escalas de motoristas de autocarros (Bus Driver Rostering) adaptado da literatura.

Os objetivos da investigação são:

- Definir um modelo de decomposição para o problema abordado, sobre o qual se possa usar o método de geração de colunas;
- Gerar um espaço de pesquisa com qualidade (com quantidade e diversidade de soluções parciais);
- Propor um novo algoritmo evolutivo adaptado à representação de soluções baseadas nas soluções parciais obtidas pelos subproblemas do modelo de decomposição durante a geração de colunas;
- Desenvolver o algoritmo híbrido que combina a utilização do método de geração de colunas, para a criação do espaço de pesquisa, com a pesquisa de soluções inteiras através de meta-heurísticas, dotado de opções de configuração que permitam parametrizar e selecionar os componentes a utilizar;

- Criar os componentes e interfaces necessários para que outras combinações (de problemas e/ou meta-heurísticas) possam facilmente ser implementadas, nomeadamente outras meta-heurísticas baseadas em populações;
- Obter soluções de qualidade para as instâncias do problema investigado;
- Obter uma configuração em que o algoritmo proposto seja competitivo na comparação com outros métodos existentes para solucionar o problema.

Esta tese apresenta as contribuições que vão de encontro aos objetivos propostos.

Considerando o problema de escalas de motoristas, a tese começa por apresentar o modelo de decomposição proposto para o problema, o qual apresenta três modelações alternativas dos subproblemas utilizados. A primeira é resultante da aplicação da decomposição de Dantzig-Wolfe sobre o modelo compacto adaptado da literatura e as restantes utilizam uma estrutura de rede para representar o subproblema e a representação do mesmo através de um problema de satisfação de restrições.

Após a apresentação dos modelos necessários para seguir a abordagem proposta, é descrita a contribuição para a expansão da *framework* que implementa o conceito de "*search by column generation*" de modo a suportar a utilização de meta-heurísticas baseadas em populações e é apresentado o algoritmo evolutivo concebido de acordo com o conceito.

É descrita a forma como são criadas as populações iniciais para fornecer às meta-heurísticas e ainda novos grupos de operadores, característicos das meta-heurísticas baseadas em populações, concretamente operadores de seleção e operadores de variação, os quais estão disponíveis para qualquer meta-heurística e podem ser adaptados/substituídos pelos problemas que utilizem a *framework*.

Definidos os componentes essenciais para a resolução do problema, é feita a descrição completa do algoritmo. A integração do novo modelo de decomposição na *framework*, as diversas configurações disponíveis na fase da criação do espaço de pesquisa com o método de geração de colunas, outras meta-heurísticas disponíveis (e utilizadas) para explorar o espaço de

pesquisa na procura da melhor solução inteira e ainda uma proposta para a utilização de perturbações, as quais já estão previstas na *framework* original e permitem expandir a pesquisa através de ciclos adicionais de geração de colunas (com restrições adicionais) e pesquisa.

Para melhorar o desempenho do método clássico de geração de colunas e para permitir encontrar soluções inteiras de maior qualidade no espaço de pesquisa, são apresentadas na tese duas contribuições adicionais: uma heurística para resolver os subproblemas de um modo mais eficiente e um operador de reparação das soluções globais utilizadas na fase de pesquisa.

A heurística proposta para a resolução dos subproblemas, mesmo que obtendo soluções que não são ótimas, pode ser utilizada para resolver os subproblemas de modo independente, como acontece quando é utilizado um método exato, ou resolver todos os subproblemas de um modo agregado de modo a obter soluções complementares entre si (as quais podem ser combinadas sem haver repetição de tarefas atribuídas).

O novo operador proposto e disponibilizado às heurísticas que exploram o espaço de soluções obtidas pela geração de colunas, no processo de reparação das soluções globais cria novas soluções para os subproblemas, se as mesmas não existirem já, caso essa solução (do subproblema) seja necessária para melhorar uma solução global.

A tese apresenta um extenso conjunto de testes computacionais realizados nas várias etapas do algoritmo desenvolvido utilizando um conjunto de trinta e duas instâncias do problema construídas considerando regras de um problema real.

O primeiro conjunto de testes é realizado na avaliação das diversas configurações alternativas para o método de geração de colunas. São testados os três modelos de decomposição (as três representações dos subproblemas) com a geração de colunas clássica, a utilização da heurística na resolução dos subproblemas de modo individual e agregado, totalizando doze configurações do algoritmo cujos resultados são divulgados nesta tese.

Na avaliação do espaço de pesquisa selecionado e para comparar o algoritmo evolutivo com outras meta-heurísticas foram realizados testes computacionais adicionais. Uma primeira avaliação do espaço de pesquisa foi feita utilizando a meta-heurística MIPSearch, que consiste em tornar as

variáveis do problema mestre restrito em variáveis inteiras e resolver o problema resultante com um algoritmo exato. A versão básica do algoritmo evolutivo proposto foi comparada com duas meta-heurísticas de solução única (*Variable Neighborhood Search* e *Simulated Annealing*) já disponíveis na *framework*.

Os resultados dos testes anteriores foram um impulsionador para o desenvolvimento do operador de reparação que foi integrado na versão final do algoritmo evolutivo. O algoritmo final foi testado com duas configurações, cuja diferença é essencialmente na dimensão e forma de gerar a população, o que tem um impacto significativo no tempo total de pesquisa. Os resultados obtidos pelo algoritmo final são comparados com os resultados obtidos resolvendo o modelo compacto do problema utilizando um método exato (*branch-and-cut*) implementado numa ferramenta comercial (CPLEX). Entre os testes computacionais são ainda apresentados os testes realizados com a utilização de perturbações para realização de múltiplos ciclos de geração de colunas e pesquisa.

Os resultados dos testes computacionais revelam a importância da utilização das heurísticas propostas. Na fase da geração de colunas, a configuração que utiliza a heurística que resolve os diversos subproblems em simultâneo foi a que obteve melhores resultados e foi selecionada para a criação do espaço de soluções para pesquisa. Os testes com as meta-heurísticas revelaram a dificuldade em obter soluções de qualidade no espaço de pesquisa original, tendo sido integrado o operador de reparação que permitiu colmatar essas dificuldades. Os resultados do algoritmo final foram comparados com os resultados da resolução do modelo compacto do problema com um *solver* comercial, revelando que as soluções obtidas pelo algoritmo proposto têm qualidade, em geral e, em muitas das maiores instâncias, os resultados foram claramente melhores.

O resultado da investigação descrita nesta tese permite responder afirmativamente à questão inicial, ou seja, desenvolveu-se um algoritmo híbrido que combina o método de geração de colunas com meta-heurísticas, concretamente um algoritmo evolutivo, o qual é eficaz na obtenção de soluções de qualidade para problemas de *Rostering* num intervalo de tempo razoável. Realça-se que na resolução de instâncias obtidas a partir de dados reais e com dimensão mais elevada, a abordagem proposta permitiu obter

soluções melhores do que as obtidas por um *solver* comercial. Considerando ainda os objetivos iniciais, destaca-se que o trabalho desenvolvido fornece uma base que permite que com pouco trabalho adicional se possa aplicar o mesmo algoritmo a outros problemas, de rostering ou não, ou testar outras configurações do algoritmo, nomeadamente na utilização de outras meta-heurísticas na fase de pesquisa.

Palavras-chave: Rostering; Meta-heurísticas; Algoritmos Evolutivos; Geração de Colunas;

Acknowledgements

I would like to thank my advisors, Professors Ana Respício and Filipe Alvelos, for been with me in this journey. I express my gratitude to Ana Respício, who accepted me as PhD student without knowing me, for providing me a challenging problem to address, for all the guidance she gave me and the knowledge shared through these years and for been patient when I was unable to be totally focused in the work. I am also grateful to Filipe Alvelos, who is part of my research career since the beginning and was available to continue advising me in the PhD (despite the distance), for giving me the opportunity to collaborate in the SearchCol project and use the framework in this research and for the guidance and valuable feedback.

I thank my colleagues for the support and encouragement, with a special reference to Graça Costa for the feedback on the papers and thesis sections I asked her to read.

I am especially grateful to my wife, Mara, and my son, Rodrigo, for been supportive when I needed, for been comprehensive in the moments I was absent and for providing me the motivation to continue working.

I thank all the institutions who funded this research. Instituto Politécnico de Setúbal, which funded the grant obtained in the program "PROTEC - Programa de apoio à formação avançada de docentes do Ensino Superior Politécnico" (SFRH/PROTEC/67405/2010), and FCT (Fundação para a Ciência e a Tecnologia) through projects PTDC/EIA-EIA/100645/2008, UID/CEC/00319/2013 and UID/MAT/04561/2013.

Contents

List of Figures

List of Tables

List of Abbreviations

BDRP Bus Driver Rostering Problem

CG Column Generation

CP Constraint Programming

CSP Constraint Satisfaction Problem

DWD Dantzig-Wolfe Decomposition

EA Evolutionary Algorithms

GA Genetic Algorithms

IP Integer Program

LP Linear Programming

LS Local Search

MAS Multi-Agent Systems

MH Metaheuristics

MIP Mixed-Integer Programming

MP Master Problem

NFL No Free Lunch

RMP Restricted Master Problem

SA Simulated Annealing

VNS Variable Neighbourhood Search

1 Introduction

Personnel scheduling is the process used by an organisation to construct work timetables for its staff to assure that its services are provided with the correct staff (quantities and categories) during its operation (Ernst, Jiang, Krishnamoorthy, & Sier, 2004b). Rostering is a special case of personnel scheduling which involves determining the specific staff to perform each service. This problem arises in a wide diversity of areas, especially in services continuously available like healthcare, transportation, call-centres, etc, where the existence of shifts is frequent and workers do not have a fixed schedule. The diversity of personnel scheduling problems, rostering included, their characterization, application areas and solution techniques employed to solve them are detailed in the annotated bibliography survey (Ernst, Jiang, Krishnamoorthy, Owens, & Sier, 2004a) which considers around 700 papers.

Rostering problems are known to be hard to solve and still have open research questions, as confirmed by the frequent new contributions related to rostering in the literature. A rostering problem has different versions, depending on the area of application, because distinct constraints can be considered or different objectives pursued.

Many relevant combinatorial optimization problems, as the rostering ones, are computationally intractable. This means that the algorithms needed to solve them would run in exponential time, resulting that computing the optimal solution for one of such problems might not be possible even without computational time constraints. Formally these problems are classified as NP-hard and with solution algorithms that run in polynomial time still unknown, we have to make use of approximate algorithms. Since real life instances of combinatorial optimization problems are typically of large size, solutions to those instances are usually obtained using heuristics.

Besides the complexity of the problems, the adoption of alternative methods to tackle optimization problems is supported by the No Free Lunch theorem

(NFL) (Wolpert & Macready, 1997). According to the NFL theorem, the average performance across all possible problems is the same for all algorithms. Comparing two algorithms, there are as many problems for which the first algorithm performs better than the second one as for which the reverse is true.

The above implies that for each problem there are algorithms that perform better than others and the "results also indicate the importance of incorporating problem specific knowledge into the behaviour of the algorithms" (Wolpert & Macready, 1997). This suggests that the knowledge about the problems to solve is important, not only to select the algorithms to use, but also to make use of that knowledge in the algorithm design. The use of approximate, i.e., heuristic methods meets these requirements, because they can be tailored to the characteristics of the problem or even make use of specific algorithms (through hybridization, for example).

Attending to the NFL theorem we can also consider that for complex optimization problems there is the need to combine two or more methods, exact or not, to apply where they perform better. This research path, the combination of exact and heuristic methods, is already followed by several authors in different problems (Blum, Puchinger, Raidl, & Roli, 2011; Dumitrescu & Stützle, 2003; Puchinger & Raidl, 2005).

In this thesis, we address the Bus Driver Rostering Problem (BDRP), a specific rostering problem, which shares most of the characteristics of other rostering problems but includes the rules that control the buses service. It is an important area of application of rostering, where many instances in the literature still do not have known optimal solutions. This particular problem is classified as NP-hard, as proved in (Respício, Moz, & Pato, 2013). Consequently, non-exact methods are the preferred to obtain good solutions in reasonable time.

1.1 Objectives

The main objective of the research performed for this thesis is to study the application of a new combination of existing methods to address rostering problems. Concretely, the combination of the column generation method with an evolutionary algorithm. The approach is applied to a rostering

problem but intents to be easily replicable in other decomposable combinatorial optimization problems.

Following the NFL theorem, we evaluate the application of this new algorithm combining an exact method and a metaheuristic to solve a hard optimization problem to which the exact methods are unable to find optimal solutions in reasonable time.

The research question we want to answer is: *Is the hybridization of the column generation method with metaheuristics effective for attaining good quality solutions for rostering problems in reasonable time?*

The proposed approach, following the concept of search by column generation proposed in (Alvelos, de Sousa, & Santos, 2010), addresses the bus driver rostering problem adapted from (Moz, Respício, & Pato, 2009) to which a new decomposition model is proposed in the thesis.

The main objectives of the research are:

- Define a decomposition model for the bus driver rostering problem solvable by using the column generation method;
- Obtain a good quality search space with a sufficient quantity and diversity of partial solutions resulting from the use of the column generation method;
- Define a new evolutionary algorithm adapted to the solutions representation based on partial solutions which are subproblem solutions obtained by the column generation;
- Develop the complete hybrid algorithm combining the column generation stage and the metaheuristic search with the necessary configuration options allowing the user to define all the parameters and components to be used;
- Provide a skeleton of the algorithm components allowing the development of new combinations with distinct metaheuristics, particularly, other population-based metaheuristics;
- Obtain good quality solutions for the BDRP instances in study;
- Reach an algorithm configuration which can be competitive in comparison with other existing methods to solve the problem in study.

1.2 Contributions and Publications

To address the problem, we developed a new algorithm following the concept of search by column generation (Alvelos et al., 2010). The algorithm is a sequential combination of the use of column generation and a new evolutionary algorithm.

To allow the use of the column generation, a new decomposition model for the BDRP is proposed and, after the optimization of the decomposition model, the evolutionary algorithm explores the set of feasible schedules resulting from the subproblems solution to obtain complete and optimized rosters.

The column generation method is a general method which can be applied to any decomposition model and the evolutionary algorithm was also designed to be used with any other problem, since the solution representation is respected.

This thesis presents the research on the addressed BDRP and, therefore, includes some additional contributions which result from the difficulties inherent to the problem and decisions made during the research.

We present the models to represent the BDRP, based on distinct subproblem formulations, new solution methods for the subproblems optimization, developed to improve the performance. The evolutionary algorithm based on the concept of search by column generation and the new repair operator which allows to obtain new subproblem solutions from within the metaheuristics. The complete algorithm, including multiple configurations on the column generation stage and the alternative metaheuristics to search for a global integer solution is also described.

Extensive computational tests compare the column generation configurations in the definition of the search space to be explore by the metaheuristics. The basic version of the proposed evolutionary algorithm is compared with three single-solution metaheuristics in the search of optimized rosters. The improved evolutionary algorithm, integrating the proposed operator to repair infeasible solutions, is tested in two configurations and the quality of the solutions is assessed. The results obtained are compared with the results of solving the problem in the original

compact formulation with a commercial solver. In the computational tests, the usage of perturbations is also tested, which is an approach proposed by the original concept followed by our current research.

The research focus on the bus driver rostering problem but a literature review on the wide literature existing for rostering problems, considering different areas of application and solution methods, is presented.

The new decomposition model for the BDRP and the evolutionary algorithm based on column generation solutions resulted in the first publication of this research.

- Barbosa, V., Respício, A., & Alvelos, F. (2013). *A Hybrid Metaheuristic for the Bus Driver Rostering Problem*. In Proceedings of the 2nd International Conference on Operations Research and Enterprise Systems (pp. 32-42). Barcelona: SCITEPRESS.

An improved version of the evolutionary algorithm with the addition of a local search procedure and elitism strategy was presented in:

- Barbosa, V., Respício, A., & Alvelos, F. (2013). *Genetic Algorithms for the SearchCol++ framework: application to drivers' rostering*. Paper presented at the IO2013 - XVI Congresso da Associação Portuguesa de Investigação Operacional, Bragança

To evaluate our evolutionary algorithm, the first population-based metaheuristic implemented in the framework was compared with other single-solution metaheuristics in the exploration of the search space:

- Barbosa, V., Respício, A., & Alvelos, F. (2015). *Comparing Hybrid Metaheuristics for the Bus Driver Rostering Problem*. In R. Neves-Silva, L. C. Jain, & R. J. Howlett (Eds.), Intelligent Decision Technologies (Vol. 39, pp. 43-53): Springer International Publishing.

To improve the column generation performance, new configurations and subproblem solution methods were developed, particularly a heuristic to solve the subproblems independently or in aggregated way. The heuristics and preliminary tests were presented in:

- Barbosa, V., Respício, A., & Alvelos, F. (2015). *A Column Generation Based Heuristic for a Bus Driver Rostering Problem*. In F. Pereira, P. Machado, E. Costa, & A. Cardoso (Eds.), Progress in Artificial Intelligence (Vol. 9273, pp. 143-156): Springer International Publishing.

The use of perturbations with short column generation cycles was tested in:

- Barbosa, V., Alvelos, F., & Respício, A. (2016). Bus Driver *Rostering by Column Generation Metaheuristics*. In J. R. Fonseca, G.-W. Weber, & J. Telhada (Eds.), Computational Management Science: State of the Art 2014 (pp. 225-231): Springer International Publishing.

A new repair operator was integrated in the EA, and in the framework, which acts as a column generator since the subproblem solutions it creates to repair a roster are included as new columns and consequently expand the search space. The repair operator was presented in:

- Barbosa, V., Respicio, A., & Alvelos, F. (2016). *A Repair Operator for Global Solutions of Decomposable Problems*. In D. Pearce & H. S. Pinto (Eds.), Proceedings of the Eighth European Starting Ai Researcher Symposium (Vol. 284, pp. 143-154).

The global algorithm and part of the extensive computational tests included in this thesis were included in a paper with title *Bus driver rostering by an evolutionary algorithm based on column generation* submitted to the journal International Transactions in Operational Research.

1.3 Organization

This thesis is organized as follows.

In Chapter 2, a literature review is presented introducing the general personnel scheduling and rostering, highlighting the characteristics that cause the large diversity of variants inside the main problem. The second chapter section reviews the solution methods found in the literature to address rostering problems, distributing the works using exact methods, approximation methods and focusing on the use of hybrid approaches

combining exact and heuristic methods. A last section reviews the works on the BDRP in particular.

The Bus Driver Rostering Problem in study is defined in Chapter 3 followed by the presentation of the models to represent the problem which definition are part of our research. A compact model adapted from the literature is presented as well as three new decomposition models for which distinction is the definition of the pricing problem (subproblem).

Chapter 4 presents the population-based search by column generation, an extension of the base framework which is introduced in the first section of the chapter. Next, the extensions to the base framework to be adapted to population-based metaheuristics are described and finally the developed evolutionary algorithm is presented, detailing all the relevant components and particularly the repair operator which was developed and integrated in the framework to be available to all the metaheuristics.

The use of algorithms based on search by column generation to solve the BDRP is described in Chapter 5. The first section shows how the models are integrated in the framework. The next section introduces the operators tailored for the problem. The third section presents the most relevant developments in the column generation stage, which is where the search space is defined. The fourth section of the chapter introduces four single-solution metaheuristics used in our research in the evaluation of the search space and in a comparison with our evolutionary algorithm. The fifth section describes the concept of perturbations, proposed in the original framework, and presents a new perturbation generator designed to the BDRP. The chapter ends with an outline of the algorithms developed.

Chapter 6 is dedicated to the presentation and discussion of computational tests. It starts with the presentation of the test instances of the BDRP and the hardware and software used in all the tests. The first set of tests is done in the column generation stage, comparing the multiple configurations and solution methods. A second set of tests compare our basic evolutionary algorithm with the single-solution metaheuristics in the search space selected in the previous tests. The tests with our evolutionary algorithm are repeated using the repair operator with two configurations and then the best results found are compared with the results obtained by a commercial solver

solving the original compact model of the problem. A last set of tests evaluates the proposed perturbations generator.

The last chapter emphasises the contributions and achievements from our research and provides an answer to our research question.

2 Literature review

This chapter presents the literature review completed during our research. The first section presents the literature related to personnel scheduling or rostering in general, detailing the distinct personnel characteristics, decisions types, types of constraints and objectives considered by the research community. The second section focus on the distinct types of methods found in the literature to address rostering problems and the last section focus specifically on the bus driver rostering.

2.1 Personnel Scheduling and Rostering

Personnel scheduling or rostering consists in defining the "work-schedule" for each of the workers in a company, or a subset of those, for a given period. Some authors, such as Ernst et al. (2004b), do not distinguish personnel scheduling from rostering, assuming that the work to distribute does not differ and it is not necessary to identify specifically the worker to whom each piece of work should be assigned. For example, when one needs to optimise the number of workers in each shift without identifying the concrete workers. Rostering arises, whenever it is necessary to identify the worker performing each work piece. A roster is a plan including the work-schedules for all workers. A work-schedule defines, for each day of the rostering period, if the worker is assigned to work or has a day-off and, in the first case, to which daily duty/shift.

The rostering problem arises because the company usually has diverse duties to assign on each day, sometimes needing particular skills, and on the other hand, the labour and company rules (days-off, rest time, etc.) restricts the blind assignment of duties to workers. The particular constraints related to company and labour rules make the problem more complex than an assignment problem (which has no constraints besides the assignment constraints), which can be solved in polynomial time by the Hungarian algorithm (Kuhn, 1955) or even more complex than the generalized assignment problem (Ross & Soland, 1975), known to be NP-hard.

Rostering is addressed in many types of businesses, as surveyed in (Ernst et al., 2004a) and more recently in (Van den Bergh, Beliën, De Bruecker, Demeulemeester, & De Boeck, 2013). Most of the research in personnel scheduling or rostering is applied in the services and transportation areas. In addition to these two main areas of application, there are also plenty of works where the area of application is undefined and some publications related to manufacturing, retail and military areas. In the services, nurse rostering is the problem which has received more attention from the researchers. In a recent literature review (Van den Bergh et al., 2013), more than 60 papers were assigned to the nurse application area, as in (Aickelin & White, 2004; Bai, Burke, Kendall, Jingpeng, & McCollum, 2010; Burke, De Causmaecker, Berghe, & Van Landeghem, 2004; Moz & Pato, 2007). There is also some research around other health care services with less impact (Brunner & Edenharter, 2011; Carter & Lapierre, 2001; Gendreau et al., 2007). Following the health care area, the problem arises in call-centres and emergency services. In the transportation area, the airline crew rostering (Kohl & Karisch, 2004; Lučić & Teodorović, 2007) is the problem receiving more attention, followed by railway (Jütte & Thonemann, 2012; Lezaun, Pérez, & Sáinz de la Maza, 2007) and buses (Moz et al., 2009; Nurmi, Kyngäs, & Post, 2011; Respício et al., 2013; Xie, Kliewer, & Suhl, 2012).

Due to the diversified application areas, the rostering process has diverse aspects to consider. Ernst et al. (2004b) identify six stages which can be part of the rostering process.

The first stage, demand modelling, has as main objective to know, for the rostering period, what the staff needs are, in each day and in each distinct period of the day. This is one of the characteristics which can vary depending on the application area. The *task-based demand* defines the demand where a list of individual duties with characteristics (time-window to be executed, duration, staff needed skill, etc.) exists and each of those duties defines the work-day of a worker. This type of demand is frequent in transport applications where the duties are built by linking trips and rest times. Another type of demand, common in nurse rostering, is the *shift-based demand*, where the number of staff for each shift is predefined. When

the demand is predicted by forecasting, it is defined as *flexible demand*. This type of demand arises in services where a high oscillation exists in the number of staff needed in the different hours of the day.

Days-off scheduling is the stage defining how the rest days are mixed with the labour days. Depending on the application, different rules may apply.

The shift scheduling stage consists in defining which shifts to use, defining the start and end times and the number of staff needed. It is used in the models of flexible demand and task demand.

The construction of a line of work, or schedule, involves the selection of a shift or a day-off for each day of the rostering period. It needs to consider restrictions on sequences of shift types, number of consecutive days without rest, etc. The different lines of work defined, altogether, need to cover all the demand. This stage is identified as the line of work construction.

Task assignment is another stage where the tasks are assigned to shifts and/or to lines of work, considering the staff skills needed and the start and end times of the tasks and shifts.

The last stage is the staff assignment. The roster is completed when, for each of the workers, a line of work is selected and those lines of work cover all the demand. This is achieved by assigning lines of work or schedules to individual workers.

Dorne (2008) describes the personnel shift scheduling and rostering process with three main blocks: the first is *staffing*, the second, *shift and roster design*, and the last, *shift and roster allocation*. The staffing block corresponds to the demand modelling in (Ernst et al., 2004b). The shift and roster design includes all the processes used to build the shifts and the complete schedules/lines of work, corresponding to the days off scheduling, shift scheduling and line of work construction stages from (Ernst et al., 2004b).

Ernst et al. (2004b) also highlight the duty generation process which is made when the tasks to assign need to be grouped to fulfil a normal work day or shift. The duty generation involves the creation of a set of feasible duties, composed by sequences of small tasks, and then a selection of a subset of those duties covering all the tasks. In the transportation applications, the

duty generation is commonly used to aggregate multiple trips and rest times (due to labour rules) that at the end are assigned to a single driver as a single task/duty.

Inside the schedules (or lines of work) construction, as presented in (Ernst et al., 2004b), cyclic and acyclic rosters (Xie & Suhl, 2015) can be used depending on the application area. Cyclic rosters classify rosters where all employees, or groups, are assigned to the same schedule differing only by the start day. In acyclic roster, each employee has an individual schedule different from others. In some applications, particularly in nurse rostering, stints are used to define the format of the admissible sequences of shifts regarding rules like maximum number of days without rest, incompatible sequences of shift (night/early), etc. Stints are shift schedules patterns. When using stints, it is possible to build cyclic and acyclic rosters by joining sequences of stints to complete each schedule. As presented in (Dorne, 2008), the design of schedules and/or rosters can be restricted by working hours (by day, week, month or other rostering period), rest period (minimum time between shifts/tasks), days-off, shift compatibility, individual preferences, skill level, service level, etc.

Besides the rostering process modules identified in (Ernst et al., 2004b) or the blocks from (Dorne, 2008), many other aspects are considered in the distinct application areas. In (Van den Bergh et al., 2013), from a selection of papers published since 2004, multiple tables are used to identify the characteristics of the problems addressed in each paper and to categorize them into four main classification fields: (1) personnel characteristics, decision delineation and shifts definition; (2) constraints, performance measures and flexibility; (3) solution method and uncertainty incorporation; and (4) application area and applicability of research.

In the next sections, some of the more relevant characteristics that distinguish the rostering problems in the literature are presented.

2.1.1 Personnel Characteristics and Decision Types

One of the relevant aspects to consider when addressing a rostering problem is the available personnel to include in the roster. Each individual worker can be distinguished by the contract type (full-time, part-time or interim), by

the skills or seniority. Most of the research papers about rostering problems consider full-time contract personnel, followed by much shorter number with part-time contract and only some considering casual workers. The skills distinction is usual in nurse rostering (Burke et al., 2004), with distinct nurse categories are still considered in recent research papers (Awadallah, Bolaji, & Al-Betar, 2015; Kuo, Leung, & Yano, 2014). De Bruecker, Van den Bergh, Beliën, and Demeulemeester (2015) present a recent review and classification of the literature regarding workforce planning problems incorporating skills and/or categories.

Another classification involving the personnel characteristics is related with the need to schedule individual employees or teams. Most of the problems from the recent literature address individual scheduling, but the use of crews (teams of employees) is usual in the transportation area (Caprara, Toth, Vigo, & Fischetti, 1998; Nishi, Sugiyama, & Inuiguchi, 2014; Xie & Suhl, 2015).

The decision types observed in the literature are also diversified. As stated before, the personnel scheduling and rostering can involve different stages, and each stage involves distinct decisions on aspects to consider in each scenario. Besides the distinction on the existence of teams or not, in a wide variety of application areas the direct assignment of a single duty to a worker is observed (ex. drivers) (Alfieri, Kroon, & Van De Velde, 2007; Hanafi & Kozan, 2014), while in others a number of workers with a minimum set of skills is needed for different periods of the day (e.g. nurses, workstations) (Bard & Purnomo, 2005; Sabar, Montreuil, & Frayret, 2009). A usual decision in most of the problems consists in the definition of the sequences of work over the days. Independently of the usage of shifts or not, in most of the problems the labour regulations prevent the assignment of incompatible duties/shifts to the same worker in consecutive days. In the problem addressed by Moz et al. (2009), which assigns duties to bus drivers, if a diver is assigned to a duty ending late on day *d*, an early duty cannot be assigned to the same driver on day *d+1* to comply with the minimum rest time. In the services using shifts the same occurs, defining the valid transition between shifts or using stints to set patterns of valid shift sequences to be assigned to workers (Brucker, Burke, Curtois, Qu, & Vanden Berghe, 2010). Decisions on the days-off structure/interval are also

present in the bibliography (Alfares, 2006; Costa, Jarray, & Picouleau, 2006; Kyngas & Nurmi, 2011).

2.1.2 Constraints and Objectives

Constraints are used to include the business rules (labour law and organisational rules) into the problem definition. Since the personnel scheduling arises in distinct areas, in order to model particularities of the area of the specific problem in study, an extensive diversity of constraints is observed in the models presented in the literature. Globally, the constraints are classified as hard or soft constraints, whether they are mandatory or whether they only define preferences, respectively. In the later, they are usually associated with penalties on the objective function to distinguish the solutions that have more constraint violations.

The type of constraints present in almost all personnel scheduling problems are the coverage constraints. They are responsible for assuring that the necessary employees are assigned to cover the demand for the complete rostering period, which is the primary objective of most of the problems. In (Van den Bergh et al., 2013), 75% of the papers in review include coverage constraints defined as hard constraints, reinforcing the importance of this type of constraints in personnel scheduling problems. The use of soft coverage constraints is also relevant in the reviewed papers, but with a significantly lower percentage, allowing the minimization of the workforce.

Personnel scheduling problems are characterized by allowing, or not, the overstaffing and the understaffing. Overstaffing occurs when the number of assigned employees is higher than the demand and understaffing occurs in the opposite. Most of the problems do not allow understaffing but there is no significant difference on the ones allowing overstaffing and not. The group of problems allowing understaffing and overstaffing simultaneously is small.

In the problems where employees are distinguished by skills or categories, the constraints are used to assure a minimum number of workers with a specific skill (through hard constraints) and sometimes employees with additional skills can be assigned, penalising the solution (through soft constraints). Most of the works use hard constraints to define skill needs and

do not allow for replacement between categories/hierarchies. Smet, Bilgin, De Causmaecker, and Vanden Berghe (2014) propose a generic model including several complex problem characteristics to the nurse rostering involving skill types and shifts.

A usual type of constraints in personnel scheduling problems is related with time. Time constraints are used to define limits (minimum and/or maximum) in a diversity of time-related aspects of the schedules. Van den Bergh et al. (2013) divide the time-related constraints into 21 groups. The most common constraints are the ones used to define the maximum/minimum number of assignments (or time) to an employee during the rostering period, weeks, or other time periods (Awadallah et al., 2015; Respício et al., 2013; Souai & Teghem, 2009). Also frequent are the constraints used to define the number of days-off, the maximum time interval between consecutive days-off or the maximum consecutive working days and, also in some cases, the weekends-off or days-off on weekend days (Nurmi et al., 2011; van der Veen, Hans, Post, & Veltman, 2015; Xie et al., 2012). Constraints on the minimum rest time between duties/shifts are also used to avoid assignments that do not respect labour rules.

Some personnel scheduling problems use additional constraints to address employee's preferences (Chiaramonte, Cochran, & Caswell, 2015; Fügener, Brunner, & Podtschaske, 2015). Besides the personnel preferences, in some models the use of balance constraints arises to reduce dissimilarities between employees (to reduce differences in the number of days-off, weekends, overtime, etc.) (Ásgeirsson, 2014; Puente, Gómez, Fernández, & Priore, 2009).

In addition to constraints, the objectives pursued in the numerous papers addressing personnel scheduling problems are also very diverse. The first main difference arises from the problems where the organizations want to optimize their human resources utilization (Burke, Kendall, & Soubeiga, 2003b; Ernst et al., 2004b; Naudin, Chan, Hiroux, Zemmouri, & Weil, 2012) versus the ones where the employee's preferences and equity concerns are considered (Borndörfer et al., 2015; Fügener et al., 2015; Martin, Ouelhadj, Smet, Vanden Berghe, & Özcan, 2013; Ouelhadj, Martin, Smet, Ozcan, & Vanden Berghe, 2012). Normally, if a roster is optimized

for the company interests (most of the times with the objective to reduce costs), the employees' preferences and fairness in the work-schedules are neglected. Inversely, pursuing to consider all the employees' preferences may result in lack of personnel to cover undesired duties or shifts. To overcome these contradictory objectives, in some problems multiple objective functions are applied (Lučic & Teodorovic, 1999; Moz et al., 2009; Respício et al., 2013; Topaloglu, 2006).

One of the most objective factors included in personnel scheduling are the financial measures, which the companies usually try to reduce, since human resources costs have a significant impact on business results. In the papers included in the review from Van den Bergh et al. (2013), the majority considers personnel costs. In that group of papers, some set different costs per day (e.g. week-day and weekend-day distinction), different costs per skill or category, and overtime costs. In some papers, costs associated with hiring outsource personnel, travel costs and costs from performing tasks, are also considered.

2.2 Solution Methods

As previously stated, personnel scheduling or rostering arises in a wide diversity of business areas, with distinct formats and details. Since the problem covers several areas and is represented by different models (to include particular details of each area), together with the fact that it is hard to obtain optimal solutions, a diversity of solution methods has been proposed in the literature, as presented in the surveys already referred (Ernst et al., 2004a; Van den Bergh et al., 2013).

In the next sections, a selection of contributions to the literature using different categories of methods is presented. In each section, the selected papers are grouped by the type of method, starting with exact solution methods, followed by non-exact methods. Independent sections are included to review works using multi-agent systems and hybrid methods (integrating exact with non-exact methods).

2.2.1 Exact Methods

The literature on the rostering problem includes diverse exact solution methods. In this section we highlight some of the works using exact methods, detailing the methods used and with what purpose.

Alfares (1998) proposes a two-stage procedure to solve the days-off scheduling problem. The proposed solution uses a formula derived from the Baker lower bound (Baker, 1974) and a minimum workforce size developed by Vohra (1987), to add a new constraint to a continuous linear programming (LP) model of the problem, which solution ensures an optimal integer solution. A similar problem is solved using dynamic programming in (Elshafei & Alfares, 2008), but in this case the patterns of the days-off have associated costs which are minimized.

The column generation (CG) method (Desaulniers, Desrosiers, & Solomon, 2005) was also used in several contributions. In (Brunner & Edenharter, 2011) a mixed-integer programming (MIP) model was proposed to model the physicians demand in an hospital for one year period. The proposed model is decomposable by week and the column generation is used to obtain the optimal linear solution to the problem. The authors solve the final restricted master problem as an integer program (IP) to obtain an integer solution which is, most of the times, the optimal solution. Bard and Purnomo (2004) uses CG to schedule nurses with individual preferences. Recently, (Smet, Ernst, & Vanden Berghe, 2016) uses CG as a base for constructive heuristics to address the integrated task scheduling and personnel rostering problem. (Borndörfer, Schulz, Seidl, & Weider, 2017) also uses a column generation approach to solve the subproblems in the proposed decomposition.

CG is commonly used as part of the branch-and-price method (Wolsey, 1998), another exact method which allows to obtain the optimal integer solution to problems modeled as MIP and that can be redefined by a decomposition model. In branch-and-price, CG is used to solve each node of the search tree, generated by adding new constraints to force variables to integer values. In (Naudin et al., 2012), three mathematical models to the staff rostering problem are presented and tested with the branch-and-price method and branch-and-bound method (in the case where the model is not a

decomposition model). In (Maenhout & Vanhoucke, 2010) different branching strategies (one of the aspects with more impact in the success of the branch-and-price method) are tested in a multiple objective nurse scheduling problem.

To reduce the costs of an inter-city bus transit firm in India, (de Matta & Peters, 2009) propose a model allowing different types of drivers to the fleet so that trips with buses from a distinct category, with regard to the principal category of the driver, can be added to the driver schedule (drivers licensed to drive bigger buses can be used to drive smaller ones or others included in the license). The model is solved with branch-and-price. More examples of branch-and-price use can be found in (Beliën & Demeulemeester, 2007; Mehrotra, Murphy, & Trick, 2000).

The sequential and integrated definition of rota scheduling and duty sequencing (roster construction) models are tested using two commercial solvers in (Xie & Suhl, 2015).

The branch-and-bound method is also used to address the drivers rostering with a days-off pattern in (Mesquita, Moz, Paias, & Pato, 2015). It solves a sequence of subproblems or the global problem with some of the variables fixed, considering the value of the linear solution.

Constraint programming (Rossi, van Beek, & Walsh, 2006) is an exact method, originated from the artificial intelligence area, which guarantees to find optimal solutions for optimization problems, if they exist, or feasible solutions for constraint satisfaction problems. The use of constraint programming to solve complete rostering problems is not usual, but one application is presented in (Cheng, Lee, & Wu, 1997), however, as stated by Qu & He (2009), the time to explore the search space of large problems is high, which results in the use of constraint programming in combination with other methods or only to solve subproblems of smaller size, as also proposed in (Qu & He, 2009) and in, for instance, (Fahle et al., 2002; Sellmann, Zervoudakis, Stamatopoulos, & Fahle, 2002; Yunes, Moura, & de Souza, 2005). In (Qu & He, 2009) only weekly subproblems are solved with constraint programming and the solutions are combined by a iterative forward search to obtain complete rosters. Fahle et al. (2002) also uses constraint programming to solve the subproblems which includes additional

constraints to reduce the search. In (Sellmann et al., 2002) constraint programming is used in the subproblems of column generation and also with a heuristic tree search to find feasible rosters. In (Yunes et al., 2005) the constraint programming is used in the subproblems in a branch-and-price algorithm.

Table 1 – Personnel scheduling literature using exact methods

Method	Reference
Branch-and-bound	(Brunner & Edenharter, 2011) (Mesquita et al., 2015) (Naudin et al., 2012) (Xie & Suhl, 2015)
Branch-and-price	(Beliën & Demeulemeester, 2007) (Maenhout & Vanhoucke, 2010) (de Matta & Peters, 2009) (Mehrotra et al., 2000) (Naudin et al., 2012)
Column Generation	(Bard & Purnomo, 2004) (Borndörfer et al., 2017) (Brunner & Edenharter, 2011) (Fahle et al., 2002) (Sellmann et al., 2002) (Smet et al., 2016) (Yunes et al., 2005)
Constraint Programming	(Cheng et al., 1997) (Fahle et al., 2002) (Qu & He, 2009) (Sellmann et al., 2002) (Yunes et al., 2005)
Dynamic Programming	(Elshafei & Alfares, 2008)
Linear Programming	(Alfares, 1998)

2.2.2 Metaheuristic Approaches

The use of heuristics (Pearl, 1984; Romanycia & Pelletier, 1985), metaheuristics (Glover & Kochenberger, 2003; Talbi, 2009c) or even hyper-heuristics (Burke et al., 2003a) to approach rostering is justified since it is more important to have an approximate solution quickly, rather than having

an exact solution that takes much more time and computational resources to obtain. In (Burke et al., 2003a) it is stated that the use of heuristic methods, particularly the hyper-heuristics, that choose from a set of heuristics which one to apply in each case, is for those who need "good enough - soon enough - cheap enough" solutions to the problems.

Metaheuristics (MH), which usually refer to general heuristics that are not specific to a particular problem, can be divided in single-solution metaheuristics (Talbi, 2009b) and population based metaheuristics (Talbi, 2009a). The first group includes all the MH which work over a single solution, as is the case of local search (Johnson, Papadimitriou, & Yannakakis, 1988), simulated annealing (Kirkpatrick, Gelatt, & Vecchi, 1983), tabu search (Glover & Laguna, 1993), iterated local search (Lourenço, Martin, & Stützle, 2003), variable neighbourhood search (Mladenović & Hansen, 1997), etc. The second group, MH that use a population of solutions, includes evolutionary algorithms (including genetic algorithms) (Michalewicz & Schoenauer, 2003), scatter search (Resende, Ribeiro, Glover, & Martí, 2010), ant colony optimization (Dorigo, Birattari, & Stutzle, 2006), particle swarm optimization (Kennedy & Eberhart, 1995), etc.

In most of the single solution MH referred, different methods are used to improve an existent solution by making small changes. The way those changes are done and controlled, results in a diversity of methods.

In the next paragraphs, some works using metaheuristic methods to solve personnel scheduling and rostering problems are presented.

In (Goodman, Dowsland, & Thompson, 2009), a nurse-scheduling problem is addressed in order to produce weekly work rosters for individual hospital wards, using a Greedy Randomised Adaptive Search Procedure (GRASP) (Feo & Resende, 1995) implementation, including a knapsack algorithm in the construction phase to assure the feasibility of the schedules assigned to nurses. The proposed approach outperforms other heuristics and was able to find optimal solutions quickly and consistently.

Variable neighbourhood search (VNS) (Mladenović & Hansen, 1997) was used in (Burke, Curtois, Post, Qu, & Veltman, 2008) to improve nurse schedules created heuristically after ordering the shifts by difficulty of

assignment, and allocating them to the nurse which results in the minimal penalty on the objective function. VNS is used to improve the schedules by trying two types of moves to change the shifts assigned to nurses. One type of move consists in assigning a shift to a different nurse and testing if the penalty decreases; the other type of move consists in swapping the shifts assigned to pairs of nurses. The proposed method achieves feasible solutions in few minutes and provides a restart mechanism used to continue the search for better solutions, by removing some assignments from a solution and going back to the ordering heuristic to reassign the shifts to the available nurses. The method was compared with a commercial tool that uses genetic algorithms and, for instances with less than twenty nurses, got better results regularly. VNS was also used in (Burke, Li, & Qu, 2010) but, this time, in a hybrid approach where some constraints are considered in an integer programming model for which solutions are subsequently improved with a VNS with additional soft constraints not included in the original model.

Brucker et al. (2010) continue working with the nurse shifts as in (Burke et al., 2008), but this time defining sequences of shifts and combining those sequences on the nurses schedules. The proposed approach is again a heuristic that divides the problem in two stages: defining the sequences of shifts and, after, assigning the sequences to the nurses considering the scheduling and rostering constraints. This division reduces the problem difficulty by considering only parts of the constraints in each stage. A greedy local search (Aarts & Lenstra, 1997) is used when the schedules are built, to improve the partial rosters.

A hyper-heuristic (Burke et al., 2003a), which is a heuristic that selects heuristics to apply to a problem, was presented and tested in the nurse scheduling problem in (Burke et al., 2003b). The proposed hyper-heuristic uses a tabu-list to manage a set of low level heuristics available to improve the solution in the hyper-heuristic iterations. The heuristics tested without improving the solution are penalized to prevent selection in the next iterations. The use of hyper-heuristics is valuable because it can adapt to different problems without a specific problem tailoring. In (Burke et al., 2003b) the proposed hyper-heuristic is tested in two distinct problems by changing only the pool of heuristics available to use in the iterations. Hyper-heuristics are also used in (Bilgin, De Causmaecker, & Vanden Berghe,

2009; Bilgin, Demeester, Misir, Vancroonenburg, & Vanden Berghe, 2012) to address nurse rostering problems and patients admission in a Belgian hospital.

In (Nurmi et al., 2011), a population based local search was used to solve a real world driver rostering for a bus transit company. The method consists in the coexistence of several local search procedures that may share information between them, helping others to avoid from getting stuck in local optimum solutions. In the presented work, the authors used a greedy hill-climbing mutation (GHCM) to move the solutions to better values by changing the day in which a shift is assigned. They address a problem from a Finnish transit company where the days-off needed to be known in an annual basis. They divide the problem in the definition of days-off schedules for the annual period and, after, the shift scheduling for a month period over the previously obtained days-off schedules. The authors continued the development of the method in (Kyngas, Kyngas, & Nurmi, 2012), where it is renamed as PEAST and it is combined with a division strategy to solve large-scale staff rostering instances.

Memetic algorithms have also been used to find solutions to rostering problems. Memetic algorithms are an extension of genetic algorithms through the use of local search to improve solutions in the genetic iterations. In (Özcan, 2005) the use of memetic algorithms in nurse rostering is presented. The algorithm uses the self-adaptive violation directed hierarchical hill climbing method (VDHC), that chooses the region of the solution (hospital, department, nurse) to apply the hill climbing, and which method to use according to the selected constraint violation. In (Özcan, 2007) the previous work was extended to the use of self-generated memes, designated as multi-meme algorithms, where not only the solutions but also the parameters from the memes, are adapted through generations.

To solve the aircrew rostering problem, Lučić and Teodorović (2007) use simulated annealing, tabu-search and genetic algorithms.

Simulated annealing (Kirkpatrick et al., 1983) is an optimization method based in the metal cooling physical process (the annealing process), that prevents the solutions from getting stuck in local optimum by allowing not

only improvements but also moves to worst solutions with a low, but dynamic, probability of occurrence.

(Peng, Shen, & Li, 2015) propose a multiple objective simulated annealing (MOSA) approach to reconcile contradictory objectives in the search for non-cyclic solutions for a bus driver rostering problem.

Tabu search (Glover & Laguna, 1993) is also a method to explore the neighbourhood of the solutions, but it keeps a memory (tabu list) of recent moves that should not be visited again in following iterations. Tabu search is also used in the selection of hyper-heuristics in (Burke et al., 2003b). In (Xie, Merschformann, Kliewer, & Suhl, 2017), the metaheuristic is compared with ant colony optimization and simulated annealing to solve the personalized crew rostering problem. (Sargut, Altuntaş, & Tulazoğlu, 2017) also uses tabu search in a parallel search on a set of solutions to the multi-objective integrated acyclic crew rostering and vehicle assignment problem.

Genetic algorithms (GA) are based on the evolution of the biological species and were proposed by Holland (1992). In the optimization context (Goldberg, 1989; Reeves, 1997), the algorithm starts with a population of chromosomes representing solutions that evolve along generations. The improvement/change of the solutions from one generation to the next is achieved by applying a selection operator that selects a subset of the best individuals from the population to reproduce. The reproduction is simulated by using the crossover operator, which is a recombination operator that merges characteristics from both parents into the offspring, by taking segments of the chromosomes alternatively from both. New characteristics can be added into the offspring by the mutation operator, whose function is to change one or more gene values to new admissible values, resulting in a small change of a solution to a neighbour solution. Details about GA are presented in (Goldberg, 1989; Mitchell, 1996; Reeves, 1997)

Genetic algorithms are one of the metaheuristics with wider usage in the non-exact solution approaches found in the literature. In the nurse rostering related problems, they are used in (Moz & Pato, 2007) in a rerostering problem to obtain new rosters with small differences from an original pre-existing roster, when the roster needs to be changed due to the absence of nurses on a day. The variation operators are used to apply permutations to

nurses and tasks and the fitness function is related to the dissimilarity of the new roster from the original, selecting as best solutions those who have small changes. Aickelin and Dowsland (2000) explore the nurse rostering problem with a different GA that uses information related to the nurses' grades to divide the population in subpopulations and to define a new crossover with a fixed-point on grade-boundaries. The results obtained were improved in (Aickelin & Dowsland, 2004) where knowledge about the problem is sent to a set of decoders responsible for building the nurse schedules according to the demand and the individual preferences. The GA is used to make permutations of nurses through generations. Aickelin and White (2004) briefly describe how the GA can be used to solve nurse rostering problems and propose a method for algorithm comparison based on using traditional statistical techniques. Souai and Teghem (2009) use GA to address simultaneously the crew-pairing and rostering problem in an airline.

In (Maenhout & Vanhoucke, 2008), the different crossover operator types are compared and a new hybridization of existing crossovers is proposed in the context of nurse scheduling. The work in (Gröbner & Wilke, 2001) also uses genetic algorithms to optimize general rostering problems using a direct representation considering a day/shift/function division to assign employees. Repair operators are used to correct infeasible solutions. Results on real data show the effectiveness of the approach. The algorithm presented was included in a commercial solution. GA are also used as the main algorithm, or as a component, in rostering problems related to healthcare in (Bai et al., 2010; Frey, Hanne, & Dornberger, 2009; Puente et al., 2009).

In (Monfroglio, 1996) GA are hybridized with a greedy heuristic to optimize rosters for a railway company. A GA was also used to optimize parameters of the greedy heuristic.

Martins and Silva (2016) uses a GA in the two stages of the algorithm proposed. The same algorithm is used in the two stages, with changes on the generation of the initial population, the mutation operator and the objective function. In the first stage the number of drivers is minimized and in the second, the workload of the drivers is harmonized considering the amount of overtime and idle time assigned.

Evolutionary algorithms are used in (Moz et al., 2009) to deal with a bi-objective function on a bus driver rostering problem trying to find the equilibrium between the workers and the company interests. In (Respício et al., 2013) a computational study tests an enhanced GA to solve the bi-objective bus driver rostering problem.

Table 2 – Personnel scheduling literature using metaheuristics

Method	Reference
GRASP	(Goodman et al., 2009)
Greedy Local Search	(Brucker et al., 2010)
Genetic algorithms	(Aickelin & Dowsland, 2000) (Aickelin & Dowsland, 2004) (Aickelin & White, 2004) (Bai et al., 2010) (Frey et al., 2009) (Gröbner & Wilke, 2001) (Lučić & Teodorović, 2007) (Maenhout & Vanhoucke, 2008) (Martins & Silva, 2016) (Monfroglio, 1996) (Moz & Pato, 2007) (Moz et al., 2009) (Puente et al., 2009) (Respício et al., 2013) (Souai & Teghem, 2009)
Hyper-heuristic	(Burke et al., 2003b) (Bilgin et al., 2009) (Bilgin et al., 2012)
Memetic algorithms	(Özcan, 2005) (Özcan, 2007)
Population based Local Search	(Kyngas et al., 2012) (Nurmi et al., 2011)
Simulated Annealing	(Lučić & Teodorović, 2007) (Peng et al., 2015) (Xie et al., 2017)
Tabu Search	(Burke et al., 2003b) (Lučić & Teodorović, 2007) (Sargut et al., 2017) (Xie et al., 2017)
VNS	(Burke et al., 2008) (Burke et al., 2010)

2.2.3 Multi-Agent Systems

Multi-Agent Systems (MAS) (Wooldridge, 2009) are systems composed of autonomous agents (Wooldridge & Jennings, 1995) with individual and global goals and a set of plans available to use according to the perception of their environment.

In the existing literature related to rostering problems, the contributions using MAS to settle the problem are short and, again, most of them are focused in the nurse rostering.

Within our knowledge, the work presented in (Kaplansky & Meisels, 2007) is one of the pioneers using MAS to address a rostering problem. A system with two types of agents is defined to manage the constraints related to the assignment of the nurses to the different wards. A scheduling agent (SA) is associated to each ward with the responsibility to build the local roster and a central resource agent (CRA) ratifies global constraints sending requests to the SA when changes are needed. The SAs use a negotiation protocol between them before contacting the central agent, improving scalability by liberating the CRA from being part of all conflicts resolution. The system was implemented in a hospital to reduce the cost of transportation of the nurses between their homes and the hospital.

A coordination model was proposed in (De Causmaecker, Demeester, Berghe, & Verbeke, 2005) to address the distributed personnel scheduling through a MAS composed of three types of agents: the employee agent (EA), the department agent (DA) and an ombudsman agent (OA). The DA is responsible for building the roster for the department, and contacts the OA when unable to find a solution by sending a list of problematic shifts. The OA starts a Contract Net Protocol (CNP) (Smith, 1980) involving all DA as contractors and also uses the same protocol to select the best EA to send back the proposal to the OA. The OA grants the DA bid with lower cost and the process restarts to the next shift.

In (Lagatie, Haspeslagh, & De Causmaecker, 2009) the coordination model from (De Causmaecker et al., 2005) is used to test different negotiation protocols in a nurse rostering problem showing that it is possible to successfully solve the problem decomposition with negotiation. The results

show that the negotiation protocols have similar quality compared to the central method but the performance is improved by a very little computation time. In (Wang & Wang, 2009) another MAS is presented for the nurse self-rostering according to preferences. The proposed MAS is composed by five types of agents, including a directory facilitator and an agency manager to coordinate the contracts of part-time nurses from outside of the hospital. A structure to represent the preferences information is presented.

In (Chiaramonte & Chiaramonte, 2008) the competitive nurse rostering (CNR) was proposed which uses a competitive agent-based negotiation to address nurse preferences. The CNR was improved with the inclusion of an iterative local search inside the agents in (Chiaramonte et al., 2015) and applied on rerostering of nurses in (Chiaramonte & Caswell, 2016).

In problems closely related to rostering, MAS have also been used. In (Günther & Nissen, 2010), a MAS is compared with a particle swarm optimization metaheuristic to define daily schedules by assigning employees to workstations in different periods of the day. Shibghatullah, Eldabi, and Kuljis (2006) proposed a solution with agents to model the crew reassignment process for bus crew on day-to-day operations. In (Sabar et al., 2009), a multi-agent approach for personnel scheduling in an assembly centre with distinct workstations is presented.

Table 3 – Personnel scheduling literature using MAS

Reference
(Chiaramonte & Chiaramonte, 2008)
(Chiaramonte et al., 2015)
(Chiaramonte & Caswell, 2016)
(De Causmaecker et al., 2005)
(Günther & Nissen, 2010)
(Kaplansky & Meisels, 2007)
(Lagatie et al., 2009)
(Sabar et al., 2009)
(Shibghatullah et al., 2006)
(Wang & Wang, 2009)

2.2.4 Hybrid Approaches

This section presents hybrid approaches to rostering problems where the hybridization includes the combination of exact methods with non-exact methods (metaheuristics). In the references used in the previous sections, some of the works also combine more than a single non-exact method to address the problem, as in (Burke et al., 2008; Monfroglio, 1996; Özcan, 2007; Ruibin, Burke, Kendall, Jingpeng, & McCollum, 2010), but only contributions that combine two types of methods are reviewed here.

The combination of metaheuristics and exact algorithms is surveyed in (Puchinger & Raidl, 2005) classifying the approaches into collaborative or integrative combinations, with other subdivisions for each branch. Inside the collaborative combinations, the sequential execution of methods and the parallel or intertwined execution are divided. Inside the integrative combinations, the survey separates the integration of exact algorithms in metaheuristics from the integration of metaheuristics in exact algorithms. Concerning the sub-classification of the integration of exact algorithms in metaheuristics, two other surveys are presented in (Dumitrescu & Stützle, 2003) and (Dumitrescu & Stützle, 2010).

In the personnel scheduling or rostering literature, the combination of constraint programming (Rossi et al., 2006) with metaheuristics is frequent. In (Cipriano, Di Gaspero, & Dovier, 2006) local search is applied to improve solutions obtained by constraint programming and, also, constraint programming is used inside a hybrid local search algorithm to explore the neighbourhood of fragments of solutions. To solve a nurse rostering problem, (Qu & He, 2009) applies a two stages hybrid method. In the first stage, constraint programming is used to solve a constraint satisfaction problem, obtaining weekly rosters with high quality shift sequences. An iterative forward search is used to build complete feasible rosters with the week rosters. In the second stage, a VNS is used to improve the solution. The hybridization of constraint programming to solve rostering problems is also used in (Bourdais, Galinier, & Pesant, 2003; Li, Lim, & Rodrigues, 2003; Stølevik, Nordlander, Riise, & Frøyseth, 2011).

Table 4 – Personnel scheduling literature using hybrid approaches

Reference
(Bourdais et al., 2003)
(Cipriano et al., 2006)
(Li et al., 2003)
(Qu & He, 2009)
(Stølevik et al., 2011)

2.3 Bus Driver Rostering

This research focus on a Bus Driver Rostering Problem (BDRP). In transportation planning systems, particularly on the buses application, the drivers' rostering is the last phase of a sequence of processes including timetabling, vehicle scheduling and crew scheduling, as in (Leone, Festa, & Marchitto, 2011; Nurmi et al., 2011; Xie et al., 2012). The literature on the BDRP is short (Moz et al., 2009). In fact, most of the papers focusing the bus driver rostering address also other stages of the transportation planning system as in (Borndörfer et al., 2017; Dorne, 2008; Leone et al., 2011; Rodrigues, de Souza, & Moura, 2006), where, before the rostering phase, there exists a phase where the shift/duties are built by defining the sequence of trips and rest time of each bus and crew and only after this definition the driver is assigned. In (Wren, 1996) the distinctions and similarities between scheduling, timetabling and rostering are discussed.

Generally, the driver rostering problem assigns specific daily duties for each driver considering a particular rostering period (Ibarra-Rojas, Delgado, Giesen, & Muñoz, 2015). Two types of rosters exist: cyclic and non-cyclic rosters. In the cyclic rosters, sequences of duties are defined, and they are assigned to drivers with a distinct starting point for each driver. In the non-cyclic rosters, the sequences of duties of each driver can be totally distinct.

Examples of research on cyclic rostering are found in (Hartog, Huisman, Abbink, & Kroon, 2009; Xie et al., 2012; Xie & Suhl, 2015). In (Hartog et al., 2009), train drivers rostering is addressed and the proposed methods starts by assigning sets of duties to each group of drivers using a mixed integer programming model and after, for each group and its subset of

duties, the roster construction includes a first stage where the types of duties (including the day-off) are assigned to each day and a second stage where each specific duty is assigned to the roster. In (Xie et al., 2012), a network model integrates the shift preferences with the admissible duty sequences. The problem is solved with a commercial solver and, after, the solution is improved using simulated annealing to reduce the maximum overtime. In (Xie & Suhl, 2015), a multi-commodity network flow model is proposed to address the cyclic and non-cyclic crew rostering integrating the rota scheduling and duty sequencing, considering the combination of multiple objectives.

The development of non-cyclic rosters is addressed in (Xie & Suhl, 2015), as mentioned before, and also in (Moz et al., 2009; Respício et al., 2013; Yunes et al., 2005). In (Moz et al., 2009), a bi-objective approach is used to obtain bus driver rosters minimizing the cost of the overtime labour and the number of drivers with incomplete work-schedules (less working days than the contracted). A computational study with enhanced algorithms for the problem is presented in (Respício et al., 2013). In (Yunes et al., 2005), an hybrid method combining branch-and-price and constraint logic programming is used to obtain the roster with all duties assigned. Xie et al. (2017) evaluate the use of ant colony optimization, simulated annealing and tabu search for constructing personalized monthly schedules for bus drivers.

Some rostering problems use work patterns to define the sequences of working days and days-off. A standard case is to simulate the normal week with five consecutive working days and two days-off (as in the week-end) as presented in (Alfares, 1998). The definition of the days-off pattern/location as the first stage of the rostering process is also observed. In (Nurmi et al., 2011), a population-based search is used to sequentially define the days-off schedule for an year period and the shift schedules for periods corresponding to a month. Two multi-commodity network flow models are also used in (Mesquita et al., 2015) to test an heuristic which gradually solves parts of a bus driver rostering problem with days-off pattern and use previous decisions to update the remaining of the problem.

As most rostering problems, the BDRP is a NP-Hard combinatorial optimization problem (Dorne, 2008; Respício et al., 2013), being

computationally hard to obtain optimal solutions. Considering the complexity of the problem and the computational time needed to achieve optimal solutions by using exact methods, many authors approach the problem with heuristic methods which are usually faster in the achievement of good solutions. According to (Van den Bergh et al., 2013), where all the personnel scheduling areas are considered, (mixed) integer programming is the preferred solution technique when using exact methods. Constructive heuristics are also frequently used, and genetic algorithms and tabu search are highlighted as improvement heuristics.

In the particular case of (bus) driver rostering, in (Yunes et al., 2005) the rostering stage is addressed with exact methods, resulting in an hybrid method combining two exact methods. In (Xie & Suhl, 2015), the sequential and integrated definition of rota scheduling and duty sequencing (roster construction) are tested using commercial solvers. A branch-and-bound algorithm including a heuristic to fix some variables in order to accelerate the convergence to integer solutions is presented in (Mesquita et al., 2015). In (Moz et al., 2009) and (Respício et al., 2013), genetic algorithms are used to improve rosters in order to consider the drivers and the company interests. Genetic algorithms are also proposed in (Martins & Silva, 2016) to obtain rosters minimizing the numbers of drivers and reducing the total number of accumulated overtime and idle time for each driver. (Peng et al., 2015) also pursued non-cyclic solutions for the BDRP that reconcile contradictory objectives using a multiple objective simulated annealing (MOSA) approach. Xie et al. (2017) claim to be the first paper using multiple metaheuristics (ant colony optimization, simulated annealing and tabu search), especially the ant colony optimization, to address the problem. (Sargut et al., 2017) shows a multi-objective crew rostering and vehicle assignment problem solved using a multi-objective tabu search algorithm working with a set of solutions in parallel.

2.4 Summary

In this chapter we started by reviewing the literature about personnel scheduling and rostering in general to reveal the multiplicity of characteristics considered in the different areas of application of the problem, including the types of decisions, constraints and objectives

commonly used in the problems. A section was devoted to categorizing the research papers according to the methods applied to address the problem. A main separation was done between those using exact methods and metaheuristics, but additional groups are presented with the research using multi-agent systems and hybrid approaches (combining two or more types of methods). The final section was devoted to reviewing the state of the art in the research related particularly to the Bus Driver Rostering.

3 Bus Driver Rostering Problem and Models

This chapter is devoted to the description of the bus driver rostering problem and the models adopted to represent it. At first, an integer programming compact model is presented and, then, column generation and Dantzig-Wolfe decomposition are introduced. The third section presents a first decomposition model which separates constraints and variables from the compact model related to a single driver into individual subproblems. Two additional decomposition models are presented where the change relatively to the first is the formulation of the subproblems. The subproblems are formulated using a network-based model and using constraint programming.

3.1 Problem Definition

The bus driver rostering problem arises in all buses companies that need to have the necessary drivers to assure all the service the company provides but also need to optimize each driver utilization since they correspond to a large slice of the operational costs of the company.

A roster is a plan with the information about the driver responsible for each duty of the company, considering all the duties in each day and all the days that the company operates. The definition of the duties is a problem solved in a previous stage by joining sequences of bus trips and rest times in order to cover all the trips the company offer (Nurmi et al., 2011). A roster line (or column) defines, for a particular driver, and for each day of the rostering period, which duty is assigned to him or if he has a day-off. It defines the work-schedule of the driver.

The challenge of the rostering problem is to optimize the company objectives (minimize costs, maximize fairness, etc), covering all the duties to assure the company service and respecting the company and the labour regulations in each work-schedule defined.

In the problem we address, the demand is task based, with the same duties in all the week days (Monday to Friday) and equal duties on weekend days (Saturday and Sunday). The rosters are acyclic, since the individual work-schedules can be totally distinct along the rostering period and different for each driver. In the work-schedule design, for each duty, the start time and total duration should be considered to avoid the assignment of invalid consecutive duties (according to labour rules), respect the maximum work time allowed (by week and rostering period), respect the day-off rules (maximum consecutive days without a day-off, minimum number of days-off in each week or in a particular day of the week) and consider as overtime the work realized beyond the regular work time for a single day.

The objective of the problem addressed is to minimize the total cost of the roster (cost of the drivers used and cost of the assigned duties).

3.2 A Compact Model for the BDRP

We consider a compact model for the BDRP derived from the one presented in (Moz et al., 2009) considering only one objective function to minimize the total cost of the roster.

The model considers a rostering horizon of K weeks ($7K$ days). The parameters used in the model are now presented and described:

V The set of drivers available to perform duties;

ρ^v Cost paid to driver v for each time unit of extra work, $v \epsilon V$. This cost allows the distinction of different salary categories of workers;

c Cost paid for each unit of work time (equal for all drivers) (included in this model to differentiate duties without overtime);

C Fixed cost paid for using a driver (equal for all drivers). The cost is not applied if the driver has no duties assigned during the rostering period (his schedule is filled up with consecutive days-off);

g Maximum number of consecutive days without a day-off;

T_h^w Set of duties on day h that must be assigned to a driver (this set does not include the "special" duty that represents a day-off), h=-g+1,…,0,1,…,7K;

T_h Set of duties to be assigned on day h (includes the "special" duty that represents the driver day-off), h=-g+1,…,0,1,…, 7K;

T_{ih}^{v} Set of duties which can be assigned to driver *v* ($v \epsilon V$) on day *h* if he is assigned to duty *i* on the previous day (*h-1*). Due to minimum rest periods, depending on the start-time and end-time of the duties, they are considered "early duties" and "late duties", and an early duty cannot succeed immediately a late duty, $v \epsilon V$, $i \epsilon T_{h-1}$, *h*=1,…, 7*K*;

t_{ih} Duration (in time units) of duty *i* on day *h*, $i \epsilon T_h^w$, *h*=1,…, 7*K*;

$\bar{t}$ Contractual daily work time (limit over which the work is considered overtime);

t'_{ih} Overtime units of duty *i* on day *h*, which results from max $\{0, t_{ih} - \bar{t}\}$, $i \epsilon T_h^w$, *h*=1,…, 7*K*;

b_1 Maximum total assigned work time (in time units) in each week of the rostering period;

b_2 Maximum total assigned work time (in time units) in all the rostering period;

d_s Minimum number of Sundays with day-off assigned to each driver during all the rostering period;

d_w Minimum number of days-off assigned to each driver in each week of the rostering period;

q Number of work days where work duties should be assigned (duties from T_h^w) to get a complete schedule to the driver. The remaining days of the rostering period are filled with days-off;

$e_{i0}{}^{v}$ Assumes value 1 if driver *v* was assigned to duty *i* on the last day of the previous rostering period, otherwise it has value 0, $v \epsilon V$, $i \epsilon T_h^w$;

$e_0{}^{v}$ Number of consecutive work days (without any day-off) driver *v* did after last day-off in the previous rostering period, $v \epsilon V$;

ϑ Index of the "special" duty which represents the day-off .

The model includes two sets of variables:

y_{ih}^{v} Binary decision variable representing if the duty *i* from day *h* is assigned to driver *v*, assuming the value 1 if true, 0 otherwise, $v \epsilon V$, $i \epsilon T_h$, *h*=1,…, 7*K*;

η^{v} Binary decision variable representing the use of the driver *v* in the roster. The variable assumes the value 1 if at least one work duty is assigned to driver *v*, 0 otherwise, $v \epsilon V$.

Based on these parameters and decision variables, the BDRP is represented by the compact integer programming model presented in Model 1.

Model 1: BDRP compact integer programming model

$$Min\ \sum_{v\epsilon V}\left(\sum_{h=1}^{7K}\sum_{i\in T_h^w}(\rho^v t'_{ih} + ct_{ih})y_{ih}^v + C\eta^v\right) \quad (1)$$

Subject to:

$$\sum_{v\in V} y_{ih}^v = 1\ , i \in T_h^w, h = 1, \dots, 7K, \quad (2)$$

$$\sum_{i\in T_h} y_{ih}^v = 1\ ,\ v \epsilon V, h = 1, \dots, 7K, \quad (3)$$

$$y_{i,h-1}^v + \sum_{j\in T_h \setminus T_{ih}^v} y_{jh}^v \leq 1\ ,\ v \epsilon V, i \in T_{h-1},\ h = 2, \dots, 7K, \quad (4)$$

$$e_{i0}^v + \sum_{j\in T_{1\setminus T_{i1}^v}} y_{j1}^v \leq 1\ ,\ v \epsilon V, i \in T_0, \quad (5)$$

$$\sum_{l=0}^{g}\sum_{i\in T_{h+l}^w} y_{i,h+l}^v \leq g\ ,\ v \epsilon V, h = 1, \dots, 7K - g, \quad (6)$$

$$\sum_{l=1}^{-e_0{}^v+g+1}\sum_{i\in T_l^w} y_{il}^v \leq g - e_0{}^v\ ,\ v \epsilon V, \quad (7)$$

$$\sum_{h=7(l-1)+1}^{7l} y_{\vartheta h}^v \geq d_w\ ,\ v \epsilon V, l = 1, \dots, K, \quad (8)$$

$$\sum_{l=1}^{K} y_{\vartheta,7l}^v \geq d_s\ ,\ v \epsilon V, \quad (9)$$

$$\sum_{h=7(l-1)+1}^{7l}\sum_{i\in T_h^w} t_{ih}\, y_{ih}^v \leq b_1\ ,\ v \epsilon V, l = 1, \dots, K, \quad (10)$$

$$\sum_{h=1}^{7K}\sum_{i\in T_h^w} t_{ih}\, y_{ih}^v \leq b_2\ ,\ v \epsilon V, \quad (11)$$

$$\sum_{h=1}^{7K}\sum_{i\in T_h^w} y_{ih}^v - q\eta^v \leq 0\ ,\ v \epsilon V, \quad (12)$$

$$y_{ih}^v \in \{0,1\}\ ,\ v \epsilon V, i \in T_h, h = 1, \dots, 7K. \quad (13)$$

$$\eta^v \in \{0,1\}, v \epsilon V \quad (14)$$

The objective function (1) minimizes the sum of the costs resulting from assigned duties (duration of the work assigned plus overtime) and the fixed costs of using the drivers.

Constraints (2) assure that each duty from each day is assigned to one, and only one, driver from the set of drivers.

Constraints (3) assure that each driver has one duty assigned in each day of the rostering period (which can be the "special" duty representing the day-off).

Constraints (4) and (5) prevent the assignment of incompatible sequences of duties in the schedule of a driver (avoiding the assignment of an early duty after a late duty). Constraints (5) consider the first day of the rostering period, where data from the last day from previous rostering period are needed. Constraints (4) consider the following days.

Constraints (6) and (7) prevent the assignment of work duties in more than g consecutive days (maximum number of work days without a day-off). Constraints (7) consider the initial days of the rostering period where information from the previous period is considered in the constraints.

Constraints (8) force the assignment of at least d_w days-off ("special" duty with index ϑ) in each week of the rostering period.

Constraints (9) force the assignment of at least d_s days-off on Sundays during the rostering period.

Constraints (10) prevent, in each week, the assignment of duties with a total duration exceeding b_1, the week work time limit defined by labour rules. Constraints (11) prevent the assignment of a complete schedule with a total duration exceeding b_2, the total work time limit defined contractually for the rostering period.

Constraints (12) force the binary variable η^v to be set to 1 if the number of duties assigned to driver v in the rostering period is between 1 and q (the maximum allowed); the variable is set to 0 if the driver schedule is filled with days-off (meaning that driver v is not used).

Constraints (13) and (14) define the variables y_{ih}^v and η^v, respectively, as binary variables.

3.3 Column Generation and Dantzig-Wolfe Decomposition

The column generation (CG) algorithm (see, for example, (Desaulniers et al., 2005)) is used when the compact model of a problem has too many variables, making its direct optimization unviable. When an original problem is modelled in a decomposition model using the Dantzig-Wolfe decomposition (Dantzig & Wolfe, 1960), the CG is commonly used to solve the linear programming relaxation of the restricted version of the resulting model (master problem).

In (Wilhelm, 2001), three types of column generation are described. In type I, an auxiliary problem is used to generate a large number of columns and then the restricted master problem (RMP) is solved considering only those columns to obtain the optimal solution with the corresponding subset of variables. In type II, the process becomes iterative and one or more subproblems are optimized to obtain new columns that are added in the RMP as new variables. The RMP solution guides the subproblems to obtain attractive solutions, and the algorithm iterates until no new columns are found by the subproblems. The type III CG is related with the Dantzig-Wolfe decomposition and it is distinct from the type II because the RMP solutions are convex combinations of the columns of each subproblem.

Before the presentation of the CG details, we need to understand how the Dantzig-Wolfe decomposition can be used over a compact model to obtain a decomposition model.

To demonstrate how the decomposition model is obtained, consider the general compact model of a minimization linear problem presented in Model 2.

Model 2: General linear program

$$(LP)\ \text{Min } c^{T}x \tag{15}$$

subject to:

$$Ax \geq b \tag{16}$$

$$Dx = d \tag{17}$$

$$x \geq 0 \tag{18}$$

Where *x* is a vector of size *n* of non-negative decision variables and *c* is a vector of the same size with the objective function coefficients. *A* and *D* are matrices of size *p*x*n* and *q*x*n*, respectively, with the coefficients of the constraints and *b* and *d* are column vectors, of size *p* and *q*, respectively, with the right-hand-side values of the constraints.

Applying the Dantzig-Wolfe decomposition (DWD), Model 2 problem can be reformulated with one or more subproblems defined by constraints (17) and (18), and a master problem defined by constraints (16).

In the decomposition model resulting from the DWD, the original variables *x* only exist in the subproblem(s). The master problem has new variables which represent solutions of the subproblem (extreme point *s*, where *S* is the set of all extreme points in the domain of *x*).

The master problem (MP) is presented in Model 3.

Constraint (21) is referred as convexity constraint and assures that the variables λ are convex combinations of the solutions for each subproblem. Constraints (20), which are the original constraints (16) using the new variables redefinition, are referred to as linking constraints, as they usually include variables from more than one subproblem, forcing them to stay in the master problem.

Model 3: DWD Master Problem

$$\textit{(MP)}\ \text{Min}\ \sum_{s\in S}(c^{T}x_{s})\lambda_{s} \tag{19}$$

subject to:

$$\sum_{s\in S}(Ax_{s})\lambda_{s} \geq b \tag{20}$$

$$\sum_{s\in S}\lambda_{s} = 1 \tag{21}$$

$$\lambda_{s} \geq 0, s \in S \tag{22}$$

Because the enumeration of all the extreme points of the subproblem(s) is impracticable, leads to the use of the column generation method. Column generation starts with a restricted master problem (RMP), i.e. a master problem where only a subset of the subproblem solutions ($S' \subseteq S$), are considered and, in each iteration, new columns (variables) are added to the RMP to improve the solution.

The addition of new variables is based on the reduced cost concept of linear programming duality.

Each iteration of the CG is composed of two steps:

- Solve the RMP with an algorithm to solve linear programming problems and obtain the dual solution variables values.
- Pricing: update the objective function of the subproblems with updated dual solution values and solve the subproblems to choose one or more columns from $S \setminus S'$ to enter the RMP.

In minimization problems, only subproblem solutions with a negative reduced cost are considered attractive to be included in the RMP, allowing the solution improvement. The subproblem objective is to search for the solution with the lowest reduced cost. If the subproblem is unable to find additional solutions with a negative reduced cost in the K space of extreme points, the method stops and the optimal solution of the RMP is the optimal linear solution for MP and LP.

Model 4: DWD Subproblem

$$\text{(SP) } Min\ (c^T - \omega^T A)x - \pi \tag{23}$$

subject to:

$$Dx = d \tag{24}$$

$$x \geq 0 \tag{25}$$

The subproblem resulting from the decomposition of Model 2 is presented in Model 4. The coefficients of the objective function include the vector ω^T with the values of the dual solution of the RMP for the linking constraints, and the scalar π with the dual value of the convexity constraint associated with the subproblem. If the objective function value of the optimum solution is negative, a new variable λ_s is created in the RMP with the subproblem solution.

The interaction between the RMP and the subproblems in each CG iteration is shown in Figure 1. After the optimization of the RMP to obtain the optimal linear solution, the dual solution is used to update the objective function of each subproblem. With the new objective function, each subproblem is also optimized and, if the solution obtained has a negative reduced cost, the solution is used to generate a new column, which results in a new variable λ_s.

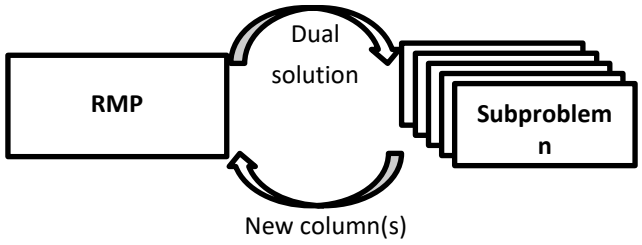

Figure 1 – Column generation cycle

As previously stated, constraints (17) from the original model (Model 2) and their respective variables are sent to one or more independent subproblems. Each subproblem solution is represented in the RMP by a new column with a unitary coefficient in the convexity constraint (21) associated to the subproblem, and with values in each of the linking constraints (20) where the subproblem variables are present (according to Ax_s).

3.4 A Decomposition Model for the BDRP

Considering the compact model presented in Section 3.2, it is easy to observe that almost all the constraints make use of variables for a single driver and only constraints (*2*) aggregate variables from all drivers. Neglecting constraints (*2*), we have one independent problem for each driver. The identified problem structure is suitable to a reformulation of the problem applying the Dantzig-Wolfe decomposition (Dantzig & Wolfe, 1960). The grouping of variables and constraint for each independent driver justifies the decomposing of the compact model "by driver".

The compact model (Model 1) can be rewritten considering the convex combination of the extreme points resulting from the subproblems' solutions, leading to a master problem that considers all possible solutions. Without loss of generality, we can assume that the set of solutions to be considered is known, thus resulting in the following restricted master problem (RMP).

Model 5: RMP formulation:

$$Min \ \sum_{v \in V} \sum_{j \epsilon J^v} p_j^v \lambda_j^v \ + \ \sum_{i \ \epsilon \ T_h^w} \sum_{h=1}^{7K} M(\delta_{ih}^+) + \sum_{v \ \epsilon \ V} M(\sigma_v^+ + \sigma_v^-) \quad (26)$$

Subject to:

$$\sum_{v \in V} \sum_{j \epsilon J^v} a_{ih}^{jv} \lambda_j^v \ + \ \delta_{ih}^+ \ \geq 1 \ , i \epsilon \ T_h^w, \ h = 1, \dots, 7K, \quad (27)$$

$$\sum_j \lambda_j^v + \ \sigma_v^+ - \sigma_v^- = 1 \ , v \epsilon \ V, \quad (28)$$

$$0 \leq \lambda_j^v \leq 1, \ j \epsilon \ J^v, v \epsilon \ V, \quad (29)$$

$$0 \leq \delta_{ih}^+ \leq 1, i \epsilon \ T_h^w, h = 1, \dots, 7K, \quad (30)$$

$$0 \leq \sigma_v^+ \leq 1, v \epsilon \ V, \quad (31)$$

$$0 \leq \sigma_v^- \leq 1, v \epsilon \ V \quad (32)$$

Where:

λ_j^v Variable associated to the schedule *j* of driver *v*;

δ_{ih}^+ Artificial variables associated to the linking constraint (for duty *i* on day *h*) to make the problem feasible until the first convex combination of extreme points is achieved by the column generation;

σ_v^+, σ_v^- Artificial variables associated to the convexity constraint (for subproblem/driver *v*) to make the problem feasible until the first convex combination of extreme points is achieved by the column generation;

J^v Set of schedules for driver *v* generated by column generation;

p_j^v Cost of the schedule *j* obtained from the subproblem of driver *v*;

a_{ih}^{jv} Assumes value 1 if duty *i* of day *h* is assigned in the schedule *j* of driver *v*;

M Very big value used to penalize the use of artificial variables in the solution of the restricted master problem.

The linking constraints (27) and convexity constraints (28) have dual variables π_{ih} and π_v, respectively, which are present in the objective function of the subproblem.

The linking constraints (27), as was the case in the corresponding constraints from the compact model (2), assure that all the duties are assigned. Since the variables on the RMP are continuous (the variables λ_j^v are relaxed), the solution of the RMP can share a duty among multiple drivers, but the sum of the columns including that duty should be 1.

An example of a RMP is presented in Figure 2. In the example, the artificial variables are omitted and the columns added by the subproblems in five iterations are presented, each one corresponding to a variable λ. The first block of rows, with labels starting with "C" are the convexity constraints identifying which subproblems is the owner of the solution used to create the column. The other blocks of rows group the linking constraints (27) for the duties in each day of the rostering period. As an example, the first column (C1) assigns every day the duty with number one, because the column has a "1" in the lines L1_1 (duty 1 from day 1), L1_2,…, L1_28.

Another example can be seen in the column C12, where the schedule represented includes taking a day-off in the first day, because there are no "ones" in all the duties of day 1, and the other duties assigned belong to the group of duties not displayed (between 3 and *n*).

	Iteration	1					2					3					4					5					
	Column	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21	C22	C23	C24	C25	
	Sp	1	2	3	…	|V|	1	2	3	…	|V|	1	2	3	…	|V|	1	2	3	…	|V|	1	2	3	…	|V|	
Constraints	C Sp1	1					1					1					1					1					=1
	C Sp2		1					1					1					1					1				=1
	C Sp3			1					1					1					1					1			=1
	C Sp…				I					I					I					I					I		=1
	C Sp|V|					1					1				1						1					1	=1
	L 1_1	1								1					1		1					1					>=1
	L 2_1		1					1				1		1				1								1	>=1
	L 3_1					1			1							1			1				1				>=1
	L …_1			1							1										1			1			>=1
	L n_1				1		1													1					1		>=1
	L 1_2	1					1			1				1			1										>=1
	L …_2		1	1		1		1	1		1	1	1					1	1		1	1		1	1	1	>=1
	L n_2				1			1							1	1				1			1				>=1
	…			…					…					…					…					…			>=1
	L 1_28	1							1			1	1		1						1				1		>=1
	L 2_28		1				1				1											1				1	>=1
	L …_28			1		1				1							1		1					1			>=1
	Ln_28				1			1						1		1		1		1			1				>=1
	Op. Sol.	0,3	0,6	0,4	0,1	0	0,1	0	0,4	0,3	0,6	0,2	0,2	0	0	0,2	0,4	0	0,2	0,6	0	0	0,2	0	0	0,2	

Figure 2 – Final RMP example

3.4.1 Subproblem

As explained in Section 3.3, the CG is used to obtain the columns needed in the RMP by solving the pricing problem (or subproblem) to identify the new columns with a positive contribution to the overall problem. Each subproblem solution corresponds to a feasible individual work-schedule for the driver associated with the subproblem.

From the application of the DWD to the compact model (Model 1), considering the variables and constraints related to a single driver, we obtain the model (Model 6) for the subproblem of a generic driver *v*. In Model 6, for simplicity of presentation, index *v* is not shown. Note that the objective function considers the dual variables of the constraints (linking and convexity) of the RMP (Model 5) according to the definition of reduced cost.

Model 6: Subproblem formulation for driver v (SPv):

$$Min \sum_{h=1}^{7K} \sum_{i \in T_h^w} (\rho t'_{ih} y_{ih} + ct_{ih} y_{ih} - \pi_{ih}\, y_{ih}) + C\eta - \pi_v \quad (33)$$

Subject to:

$$\sum_{i \in T_h} y_{ih} = 1 \ , h = 1, \dots, 7K, \quad (34)$$

$$y_{i,h-1} + \sum_{j \in T_h \setminus T_{ih}} y_{jh} \leq 1 \ , i \in T_{h-1}, h = 2, \dots, 7K, \quad (35)$$

$$e_{i0} + \sum_{j \in T_1 \setminus T_{i1}} y_{j1} \leq 1 \ , i \in T_0, \quad (36)$$

$$\sum_{l=0}^{g} \sum_{i \in T_{h+l}^w} y_{i,h+l} \leq g \ , h = 1, \dots, 7K - g, \quad (37)$$

$$\sum_{l=0}^{g-e_0+1} \sum_{i \in T_l^w} y_{il} \leq g - e_0, \quad (38)$$

$$\sum_{h=7(l-1)+1}^{7l} y_{\vartheta h} \geq d_w \ , l = 1, \dots, K, \quad (39)$$

$$\sum_{l=1}^{K} y_{\vartheta,7l} \geq d_s, \quad (40)$$

$$\sum_{h=7(l-1)+1}^{7l} \sum_{i \in T_h^w} t_{ih}\, y_{ih} \leq b_1 \ , \ l = 1, \dots, K, \quad (41)$$

$$\sum_{h=1}^{7K} \sum_{i \in T_h^w} t_{ih}\, y_{ih} \leq b_2, \quad (42)$$

$$\sum_{h=1}^{7K} \sum_{i \in T_h^w} y_{ih} - q\eta \leq 0, \quad (43)$$

$$y_{ih} \in \{0,1\} \ , i \in T_h, h = 1, \dots, 7K, \quad (44)$$

$$\eta \in \{0,1\}; \quad (45)$$

Where:

y_{ih} Binary variable representing if duty *i* from day *h* is assigned to driver associated with this subproblem, assuming the value 1 if true, 0 otherwise, $i \epsilon T_h$, *h*=1,…, 7*K*;

η Binary variable representing the use of the driver associated with this subproblem. The variable assumes the value 1 if at least one work duty is assigned to driver, 0 otherwise (schedule full of days-off);

ρ Cost paid to driver of subproblem *v* for each time unit of extra work;

π_{ih} Dual variable associated to the linking constraint of duty *i* of day *h* (constraints (27) from the RMP);

π_v Dual variable associated to the convexity constraint (constraints (28) from the RMP) inserted in the restricted master problem associated with this subproblem *v* (driver *v*);

T_{ih} Subset of T_{ih}^{v} (defined in the compact model) related to the subproblem driver *v*;

e_0 Number of consecutive work days (without day-off) the subproblem driver did after the last day-off in the previous rostering period;

e_{i0} Assumes value 1 if subproblem driver was assigned to duty *i* on the last day of the previous rostering period, otherwise it has value 0, $i \epsilon T_h^w$;

All the other parameters remain the same as in the compact model.

In each iteration of the CG, the objective function (33) is updated with the π_{ih} and π_v dual variables value from the last RMP optimization and, if the objective solution value is negative, the subproblem solution is used to generate a new column. In the new column, the value of a_{ih}^{jv} is 1 if the duty with index *i* on day *h* is included in the solution of the subproblem of driver *v* in iteration *j*. The coefficient p_j^v of the new variable in the RMP is calculated considering the original costs:

$p_j^v = \sum_{h=1}^{7K} \sum_{i \in T_h^w} (\rho t'_{ih} y_{ih} + ct_{ih} y_{ih}) + C\eta$.

3.4.2 Network Model Subproblem

Considering the decomposition model presented in Section 3.3, in this section a new decomposition model is presented where the subproblem model adopts a network structure as in other rostering models in the literature (Cappanera & Gallo, 2004; Moz & Pato, 2003; Xie & Suhl, 2015). To adopt a network structure, the duties are represented by the nodes and a transition from one duty to another (two duties in consecutive days), is represented by an arc linking the nodes. The variables are redefined accordingly, representing if an arc is used or not, to identify the sequence of duties of the driver schedule for the 7*K* days (*K* weeks).

Figure 3 shows an example of a network representing the duties (separated vertically) for each day (separated horizontally). The last duty (with no fill color) represents the day-off duty for the corresponding day. The labels of

the nodes identify the day of the duty in the first index and the duty number in the second index. The flows represent the possibilities to choose a path from a virtual start node (before the day zero) to an end node (after the last day). The cost of using each arc depends on the destination duty, since it represents to be assigned to that duty on that day, incurring in the cost.

The schedule of a driver not used is defined by a path from the start node to the end node traversing all day-off nodes. All the arcs ending in those nodes have a null cost. If a driver' schedule only includes days-off, the fixed cost from using the driver is not applied.

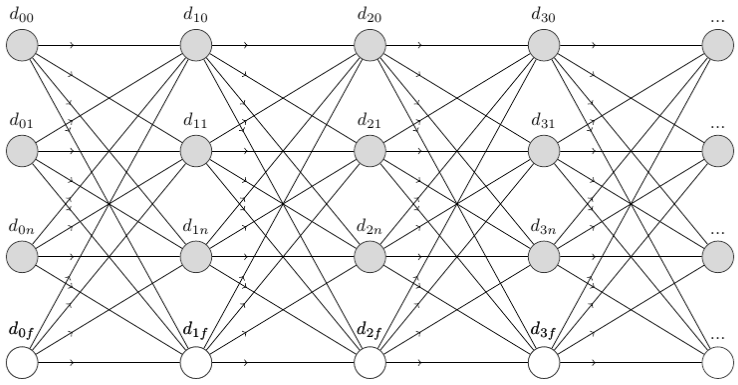

Figure 3 – Network structure example

The network structure from Figure 3 only defines the available duties and the valid sequences from the duties of a day to the ones of the next day. The additional constraints related with the maximum number of consecutive work days, number of days-off in each week and the limits on the accumulated work time still present in the new model, adapted to the new variables definition. The resulting formulation for the subproblem is described in the Model 7.

The variables used in the model to represent the arcs usage are identified by x_{ij}^{h}. It represents the variable associated with the arc which links the duty *i* from day *h-1* to the duty *j* from day *h*. Two new virtual duties (S and E) represent the source and the sink of the network. They are the only origin/destination of the arcs to/from duties of the first/last day of the rostering period. The variable η remains unchanged from the previous model.

Besides the constraints from the formulation in Model 6, which are converted to the new variables definition in constraints (50) to (60) , a set of new constraints, designated as flow conservation constraints (Ahuja, Magnanti, & Orlin, 1993), are included in the model. Constraints (49) are used to assure that the flow entering a node is equal to the flow leaving that node. Constraints (47) and (48) are responsible for, respectively, "produce" and "consume" the flow of the network, which is unitary to define a path from start to end.

Model 7: Network Model formulation for subproblem of driver *v*:

$$Min\ \sum_{h=1}^{7K}\sum_{j\in T_h^w}\sum_{i\in T_{h-1}^w}\left(\rho t'_{jh}x_{ij}^h + ct_{jh}x_{ij}^h - \pi_{ih}^L x_{ij}^h\right) + C\eta - \pi_v^C \quad (46)$$

Subject to:

$$\sum_{j\in T_1} x_{Sj} = 1 \quad (47)$$

$$\sum_{i\in T_{7K}} x_{iE} = 1 \quad (48)$$

$$\sum_{i\in T_{h-1}} x_{ij}^h - \sum_{k\in T_{h+1}} x_{jk}^{h+1} = 0\,, i \in T_{h-1}, h \in \{1,\dots,7K\} \quad (49)$$

$$\sum_{j\in T_h \setminus T_{ih}} x_{ij}^h = 0, i \in T_{h-1}, h \in \{2,\dots,7K\} \quad (50)$$

$$\sum_{j\in T_1 \setminus T_{i1}} x_{ij}^1 = 0, i \in T_0 \quad (51)$$

$$\sum_{l=0}^{g}\sum_{i\in T_{h-1+l}^w}\sum_{j\in T_{h+l}^w} x_{ij}^{h+l} \le g\,, h \in \{1,\dots,7K-g\} \quad (52)$$

$$\sum_{l=0}^{g-e_0+1}\sum_{i\in T_{l-1}^w}\sum_{i\in T_l^w} x_{ij}^l \le g - e_0 \quad (53)$$

$$\sum_{h=7(l-1)+1}^{7l}\sum_{i\in T_{h-1}} x_{i\vartheta}^h \ge d_w\,, l \in \{1,\dots,K\} \quad (54)$$

$$\sum_{l=1}^{K}\sum_{i\in T_{7l-1}} x_{i\vartheta}^h \ge d_s \quad (55)$$

$$\sum_{h=7(l-1)+1}^{7l}\sum_{i\in T_{h-1}}\sum_{j\in T_h^w} t_{jh}x_{ij}^h \le b_1\,, l \in \{1,\dots,K\} \quad (56)$$

$$\sum_{h=1}^{7K}\sum_{i\in T_{h-1}}\sum_{j\in T_h^w} t_{jh}x_{ij}^h \le b_2 \quad (57)$$

$$\sum_{h=1}^{7K}\sum_{i\in T_{h-1}}\sum_{j\in T_h^w} x_{ij}^h - q\eta \le 0 \quad (58)$$

$$x_{ij}^h \in \{0,1\}\,, i \in T_{h-1}, j \in T_h, h \in \{1,\dots,7K\} \quad (59)$$

$$\eta \in \{0,1\}; \quad (60)$$

In each iteration of the CG, the objective function (46) is updated with the π_{ih} and π_v dual variables value from the last RMP optimization, which is the same, and if the objective solution value is negative, the subproblem solution is used to generate a new column. In the new column, the value of a_{ih}^{jv} is 1 if any of the arcs ending in the node representing the duty *i* on day *h* is included in the solution of the subproblem of driver *v* in iteration *j*. The coefficient p_j^v of the new variable in the RMP is calculated considering the original costs:

$$p_j^v = \sum_{h=1}^{7\mathrm{K}} \sum_{j \in T_h^w} \sum_{i \in T_{h-1}^w} \left(\rho t'_{jh} x_{ij}^h + \mathrm{c}t_{jh} x_{ij}^h\right) + C\eta$$

3.4.3 Constraint Programming Subproblem

This section presents a new reformulation of the subproblems for the decomposition model of the BDRP. In this model, the subproblem is described as a constraint satisfaction problem (CSP) by a constraint programming (CP) model. In this new model, the binary variables which define the work-schedule of the driver are replaced by a variable for each day, with a domain of integer values corresponding to the duties (plus day-off) available in that day. The new variables definition, a new group of sets to represent the problem data and the new constraints used to formulate the subproblem as a CP model, are presented next.

Variables:

WS(days)	Variables that define which duty is assigned in each day of the work-schedule. The domain of these variables corresponds to the indexes of the duties in T_h, where *h* is the day. The index zero identifies the virtual duty representing a day-off (driver not working).
DriverUsed	Variable with domain {0,1} used to represent if the driver associated to the subproblem is used or not. The driver is not used if the WS variables are filled with the index of the day-off duty for all the days.

Sets:

CDuals	Solution value of the dual variable of the convexity constraint (28) in the RMP.
$LDuals_{h\,i}$	Solution values of the dual variables of the linking constraints (27) corresponding to all duties (i) of each day (h).
Days	All the days considered in the rostering period.
Sundays	Days corresponding to Sundays.
WkDays, *WndDays*	Indexes of the week (Wk), weekend (Wnd) days considered in the rostering period.
WkDutiesT, *WndDutiesT*	Duration of each duty in the week (or weekend) days.
WkDutiesOT, *WndDutiesOT*	Overtime included in each duty available in the week (or weekend) days.
WkWkDays	Days of the week whose successor is not a weekend day (Monday to Thursday).
WkWndDays	Days of the week whose successor is a weekend day (Fridays).
WndWndDays	Days of the weekend whose successor is a weekend day (Saturdays).
WndWkDays	Days of the weekend whose successor is a week day (Sundays).
EWkDuties	Week duties considered "early duties" according to the starting time.
forbiddenWkWk	Pairs of duties which cannot be assigned in consecutive days (late duty on day d and early duty on day d+1), considering a transition between two week days.
forbiddenWkWnd	Pairs of duties which cannot be assigned in consecutive days, considering a transition between a week day and a weekend day.
forbiddenWndWnd	Pairs of duties which cannot be assigned in consecutive days, considering a transition between two weekend days.

forbiddenWndWk	Pairs of duties which cannot be assigned in consecutive days, considering a transition between a weekend day and a week day.
d_{r0}	Number of days worked in the previous roster, by each driver, since last day-off.
Previous	Type of duty done by each driver in the last day of the previous roster (1 - Late; 0 - Other).

The subproblem formulation using constraint programming is presented in Model 8.

Model 8: CP Model formulation for subproblem of driver v:

Minimize Obj

Subject to:

$$\sum_{h=d}^{d+g}[WS_h = 0] \geq 1, d \in \{1, \dots, 7K - g\} \quad (61)$$

$$\sum_{h=1}^{g+1-d_{r0}[v]}[WS_h = 0] \geq 1 \quad (62)$$

$$\sum_{h=(w*7)+1}^{(w*7)+7}[WS_h = 0] \geq d_W,\ w \in \{0, \dots, K\} \quad (63)$$

$$\sum_{h \in Sundays}[WS_h = 0] \geq d_S \quad (64)$$

$$\sum_{h=(w*7)+1}^{(w*7)+5} WkDutiesT_{WS_h} + \sum_{h=(w*7)+6}^{(w*7)+7} WkDutiesT_{WS_h} \leq b_1, w \in \{0, \dots, K\} \quad (65)$$

$$\sum_{h \in WkDays} WkDutiesT_{WS_h} + \sum_{h \in WndDays} WkDutiesT_{WS_h} \leq b_2 \quad (66)$$

$$WS_h = a \Rightarrow WS_{h+1} \neq b, (a, b) \in forbiddenWkWk, h \in WkWkDays \quad (67)$$

$$WS_h = a \Rightarrow WS_{h+1} \neq b, (a, b) \in forbiddenWkWnd, h \in WkWndDays \quad (68)$$

$$WS_h = a \Rightarrow WS_{h+1} \neq b, (a, b) \in forbiddenWndWnd, h \in WndWndDays \quad (69)$$

$$WS_h = a \Rightarrow WS_{h+1} \neq b, (a, b) \in forbiddenWndWk, h \in WndWkDays \quad (70)$$

$$Previous_v = 1 \Rightarrow WS_0 \neq forbiddenDuty,$$

$$forbiddenDuty \in EWkDuties \quad (71)$$

$$\sum_{h \in Days}[WS_h = 0] + 7K * DriverUsed \geq 7K \quad (72)$$

$$Obj = DriverUsed * C - CDuals_v + \sum_{h \in WkDays}\left(WkDutiesT_{WS_h} + WkDutiesOT_{WS_h} * \rho - LDuals_{h\,WS_h}\right) + \sum_{h \in WndDays}\left(WndDutiesT_{WS_h} + WndDutiesOT_{WS_h} * \rho - LDuals_{h\,WS_h}\right) \quad (73)$$

The set of constraints (61) counts the number of days-off (duty with index 0) assigned in each group of $g+1$ consecutive days, making sure that at least one day-off is counted to respect the maximum number of days without a day-off. The set of constraints (62) applies for the first days of the rostering period, the information about the number of days worked in the end of the previous roster is considered to define the valid location of the first day-off.

Constraints (63) assure the counting of at least d_W days-off in each week and the constraint (64) assures the existence of at least ds days-off assigned on a Sunday.

Constraints (65) limit the sum of working-time for each week and constraint (66) sets the maximum on the total duration of the duties assigned in the entire work-schedule.

The group of constraints (67) to (70) avoids the assignment of infeasible consecutive duties (transitions from a late duty to an early duty). Constraints (71) prevent the selection of early duties in the first day if the driver did a late duty in the last day of the previous roster.

Constraint (72) forces the variable that indicates if the current driver is used to assume value 1 if the count of days-off (duty with index zero) is lower than $7K$ (the number of days of the rostering period).

The objective function to minimize is defined by (73), where the cost of the work-schedule is calculated considering the fixed cost of using the driver, when true, the cost of the worktime based on the assigned duties and the cost of the overtime considering the cost of each time unit for the driver. The dual solution value (from the RMP) from each of the linking constraint (27) corresponding to the duties included in the work-schedule is subtracted and also the dual solution value of convexity constraint (28) corresponding to the subproblem.

When this formulation is used as the CG pricing problem to obtain the new columns for the RMP, if the objective function value is negative, the subproblem solution is used to generate a column, as in the other subproblem formulations. For the column of iteration *j* generated by a solution from the subproblem of driver *v*, the value of a_{ih}^{jv} is set to 1, for each *h* in 1..7*K*, if $i = \mathrm{WS_h} - 1$, since the variable $\mathrm{WS_h}$ contains the index of the duty assigned on day *h*, but the index 0 is used to represent the day-off and only the real duties are present in the RMP.

The coefficient p_j^v of the new variable in the RMP is calculated considering the original costs:

$$p_j^v = DriverIsUsed*C + \sum_{h \in WkDays}\left(WkDutiesT_{WS_h} + WkDutiesOT_{WS_h} * \rho\right) + \sum_{h \in WndDays}\left(WndDutiesT_{WS_h} + WndDutiesOT_{WS_h} * \rho\right)$$

3.5 Summary

This chapter was devoted to describing the Bus Driver Rostering Problem we addressed as well as the models used and proposed in the research described in this thesis. We started with the problem definition and with the presentation of the compact model adapted from the literature to represent the problem in study. A section introduced the column generation method and the Dantzig-Wolfe decomposition followed by the presentation of a decomposition model proposed to the BDRP. Besides the subproblem resulting from applying the Dantzig-Wolfe decomposition to the compact model, two additional models are presented, one based on a network structure and another using constraint programming.

4 Population-Based Search by Column Generation

This chapter starts with the presentation of the concept of search by column generation, in which our research is based. The next section describes some extensions to the framework which implements the concept to allow the usage of population-based metaheuristics and the last section describes an evolutionary algorithm based on column generation.

4.1 Search by Column Generation

A general description of the theoretical concept of Search by Column Generation framework was first presented in (Alvelos et al., 2010). This framework proposes a new concept of using metaheuristic search combined with column generation to obtain approximate solutions of decomposable optimization problems. Further developments and a more detailed description of the framework concept and implementation are described in (Alvelos, Sousa, & Santos, 2013).

The purpose of the framework is to achieve good quality solutions for the complete problem reducing the time needed for a complete search for the optimal solution.

The main idea is to allow the use of diverse metaheuristics to build good solutions for a wide variety of problems. There are two conditions imposed by the framework:

1. The problems which may integrate the framework must have a decomposition integer programming model, allowing the optimization of its linear relaxation by using the column generation method;
2. The metaheuristics explore the space composed by the subproblems' solutions saved through the column generation process.

Figure 4 presents an overview of the major components of the SearchCol framework.

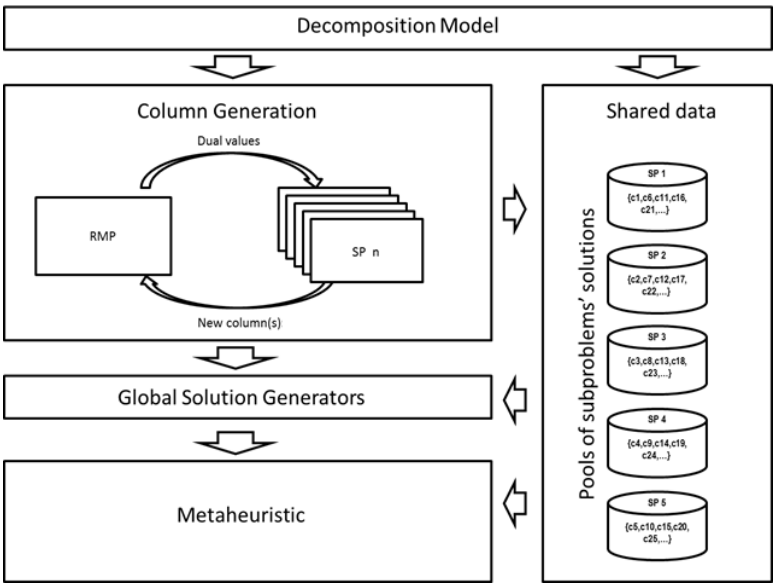

Figure 4 – SearchCol outline

After the decomposition model initialization, the column generation is used to solve the problem. The column generation method solution is an optimal linear solution composed by the convex combination of subproblems' integer solutions obtained by solving the subproblems as integer problems through the iterations.

During the column generation iterations, the subproblems need to be optimized in each iteration with updated objective function coefficients. Subproblems can be optimized using an exact optimization method, for example, a mixed integer programming model solved by a commercial solver or a method specific to the subproblem. At expenses of losing the optimality of the linear relaxation, heuristics can also be used to solve the subproblem with the purpose of obtaining solutions more efficiently.

If no heuristics are used in the subproblem and column generation runs until optimality, at the end an optimal linear solution is achieved and the pools of subproblems solutions (shared data in Figure 4) are filled with the solutions used to create each attractive column added to the restricted master problem. The pools of solutions are the source from where a solution can be picked with the certainty that it is a admissible solution for the corresponding subproblem. Picking a solution from each of the pools results in a global integer solution (for the complete problem), since all subproblems solutions

are integer. However, the solution needs to be evaluated to check if it is feasible (if the linking constraints are satisfied) and, if true, its value is computed.

The framework has a set of global solution generators to build integer solutions with the information stored during the column generation. For most of the decompositions, a global solution for the complete problem is built with a subproblem solution from each subproblem. The work done by each of the generators is similar, the difference lies in the way each subproblem solution is selected from the pools of stored solutions.

The more universal generators are those where each subproblem solution is randomly selected from each pool, with the difference that the probability of each one to be chosen can be uniform or biased by the value of the variable associated to the column generated by that solution in the last optimal solution of the RMP. The generator using the biased probability considers that if a subproblem solution belongs to the optimal linear solution of the RMP, it is probably better than others not belonging or with a lower contribution.

There is a wide diversity of generators considered in the framework, some with a stochastic behaviour, as those mentioned, and others with a deterministic behaviour (rounding up, greedy, etc.). The conceived generators and their behaviour are described in (Alvelos et al., 2013).

As a global solution obtained by the generators is not guaranteed to be a good quality solution or even a feasible one, it is evaluated according to two values (its feasibility value and infeasibility value). The feasibility value is related with the cost of the global solution and the infeasibility value is related with the number of violated linking constraints, allowing to identify if a global solution is feasible or not, and to measure how many constraints are not respected.

The function that evaluates a global solution uses the RMP to count the number of constraints for which the solution is infeasible. However, to make use of the problem knowledge or to create specific cost functions, each decomposition model can customize the evaluation function, which replaces the original one when used over global solution of that decomposition.

A global solution, or a population of those global solutions, is the starting point from which the metaheuristics are used to get a good global solution. Since a global solution is a selection of one solution from each subproblem, the search space available to each metaheuristic is the set of pools of solutions from all the subproblems from where a new solution can be selected. Each metaheuristic knows which part of the global solution corresponding to a subproblem can be replaced by any other from the pool of solutions from that subproblem.

Considering the cost of each subproblem solution (feasibility value), which is stored in the solution, the metaheuristic can use the evaluation function to obtain the global solution value (both feasibility and infeasibility values) and to identify if a change results in a better or worse global solution. Depending on the metaheuristic behaviour, the changes from better solution values to worst solution values can be accepted or rejected. Each metaheuristic implements its own strategies to explore the solution space and try to achieve the global optimum.

A sequence of consecutive runs of the column generation and the metaheuristic search constitutes an iteration of a wider search cycle. The framework allows using perturbations, which are new constraints inserted in the RMP, in order to force the generation of new columns if the wider search cycle is used and, then, to repeat the search in the resulting search space. The main algorithm is described in Figure 5.

```
1: Column generation
2: Metaheuristic search
3: REPEAT
4:     Define column generation perturbation
5:     Optimize perturbed column generation
6:     Metaheuristic search
7: UNTIL Stopping criterion fulfilled
```

Figure 5 – SearchCol algorithm (with perturbations)

If the algorithm is configured to run multiple searches, the use of perturbations is needed, otherwise the column generation cannot obtain new columns. A perturbation intends to fix (to zero or to one) a subproblem variable by adding a new constraint in the RMP. After adding the perturbation, fixing a subproblem variable to zero or one, only the

subproblem solutions, and corresponding columns, where the new constraint is satisfied will be considered by the algorithm.

For a detailed description of the perturbation concept and a presentation of the existing implementations, the reading of (Alvelos et al., 2013) is suggested. As occurs with other components of the SearchCol framework, specific perturbations can be used by defining how the variables to fix are selected and to which values they are fixed.

The SearchCol framework was used to successfully develop algorithms to multiple problems. In (Santos, de Sousa, & Alvelos, 2013) the network load balancing problem is addressed with a SearchCol algorithm where the metaheuristic used is the Greedy Randomized Adaptive Search Procedure (GRASP) with path relinking. In (Santos, de Sousa, Alvelos, & Pióro, 2013) the perturbations are included in the previous algorithm. The Forest Harvest Scheduling Problem was also addressed with an algorithm based on the framework in (Martins, Alvelos, & Constantino, 2015), the Two- and Three- Stage Bin Packing Problems in (Alvelos, Silva, & de Carvalho, 2014) and the Machine Scheduling Problem with Job Splitting in (Florêncio, Pimentel, & Alvelos, 2015).

4.2 Extensions to the SearchCol Framework

The original SearchCol framework, proposed in (Alvelos et al., 2010) and detailed in (Alvelos et al., 2013), was designed for single-solution metaheuristics, proposing a set of initial solution generators and implementations of some of the most popular single-solution metaheuristics. In this section, an extension of the SearchCol framework to allow using population-based metaheuristics is proposed. This extension relies on conception of new components and their use in a new evolutionary algorithm following the concept proposed by SearchCol.

4.2.1 Generation of Populations

The SearchCol framework relies on the use of metaheuristics to search for the best combination of subproblem solutions defining a feasible global solution which optimizes the objective function. Most of the metaheuristics need a starting solution which they try to improve by exploring the search

space. The framework allows choosing from a set of generators to obtain a global starting solution, including deterministic or stochastic behaviour in the selection of the subproblem solutions to be included.

Some of the generators are introduced in (Alvelos et al., 2013). The selection and configuration of each of the available generators is made through parameters on runtime, allowing to easily evaluate the impact of the starting solution obtained by the distinct generators in the metaheuristic search.

One of the assumptions of the SearchCol concept is that, besides the good quality of the search space resulting from the column generation, the optimum linear solution is a good pointer for the best subproblem solutions to consider. Some generators use the values of the RMP optimum linear solution to distinguish the subproblem solutions, considering that, if a subproblem solution has a high contribution to the optimum linear solution, it is also good to integrate the integer solution. The simpler case is a generator where the subproblem solutions are selected randomly from each pool but the probability of each solution to be selected is biased by the corresponding solution value in the optimum linear solution.

An evolutionary algorithm (EA) integrates the group of population-based metaheuristics , meaning that it evolves a population of individuals, contrary to other metaheuristics such as Local Search (LS), variable neighborhood search (VNS), simulated annealing (SA), etc. where a single solution is used (Talbi, 2009b). The construction of the initial population to be used by the EA is very important because it defines the base information to be explored by the EA through the generations. The initial population used by the EA should assure the equilibrium between having good quality solutions and a good diversity of solutions. If many solutions are identical, the EA will have difficulty to evolve the population.

The new population generator simply uses the already available generators (of single solutions) to create each individual to include in the population. The population generator is independent of the metaheuristic. The number of individuals created by each generator is controlled by parameters read at execution time, allowing testing different configurations easily. The sum of

the number of individuals created using all the individual generators defines the population size of the EA.

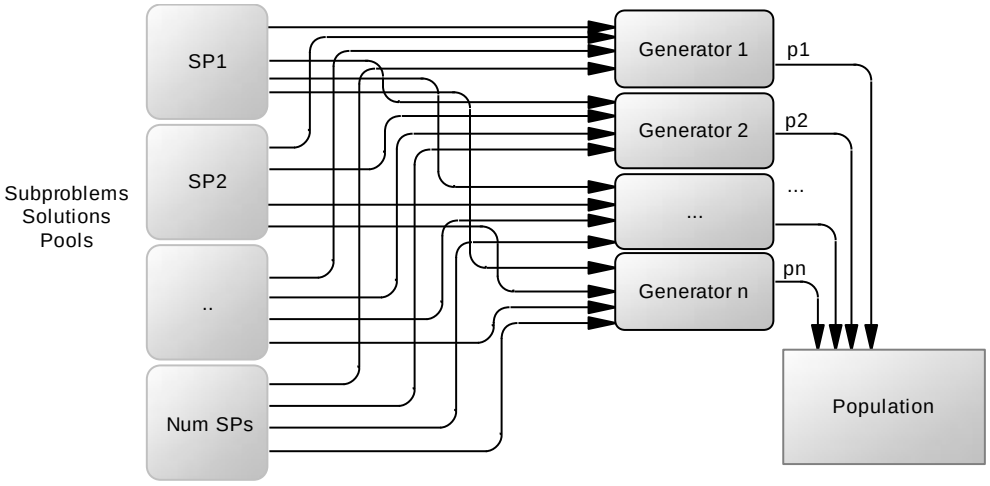

Figure 6 – Overview of the generation of an initial population

Figure 6 represents the generation of a population using *n* distinct generators. The number of global solutions obtained from each generator is controlled by the parameters p_1, p_2,… , p_n. The size of the resulting population depends on the values of these parameters and also on the generators selected, because some of the generators available in the framework use the parameter as a factor to multiply by the number of subproblems available. The idea is that the number of global solutions obtained from the generator to the population is proportional to the size of the problem (number of subproblems). The generators with a stochastic behaviour return a number of solutions equal to the number of subproblems in each execution.

4.2.2 Operators for Population-Based Search

This section describes the new operators, we included in the framework, which are needed by our EA and can be shared with new population-based metaheuristics that may integrate the framework in the future. The new operators are divided in two categories, selection and variation operators. The chosen selection operator to integrate the framework and the EA is the tournament selection and, as variation operators, two crossover operators (one and two-points) and a mutation operator were developed.

These new operators are included in the framework not only to allow modifications or the design of alternative operators by other problems using the framework, but also to allow that they can be used by diverse metaheuristics, as represented in Figure 7.

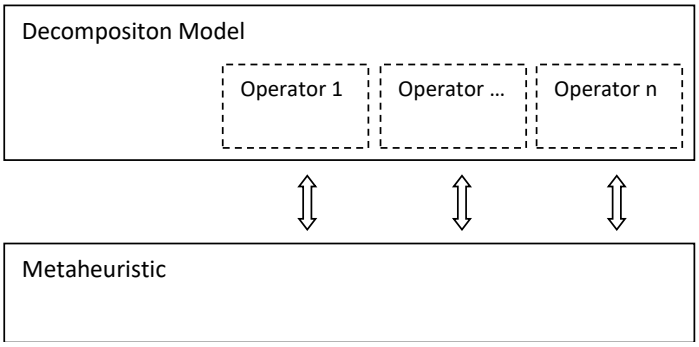

Figure 7 – Metaheuristic access to the new operators

Selection operator

Having an initial population, the EA needs a selection operator to build the mating pool. The mating pool is a selection of the best individuals (according to their fitness function values) in the current generation that will be used to generate the population of the next generation. There are different selection operators, some more simple, like the "roulette wheel", where the slice of the roulette can be proportional to the fitness value of the individual and others more computationally expensive like sigma scaling or Boltzmann selection (Mitchell, 1996).

The selection operator currently provided by the framework is the tournament selection, which is defined detached from the EA or any other metaheuristic. We choose this operator because it requires less computation effort since only the involved individuals need to be evaluated. The tournament concept is very simple, taking two or more individuals from the population, they are compared and the one with best fitness (better value in the fitness function) wins, entering the mating pool to potentially become a parent for the next generation.

Details on the solution representation and evaluation are presented in sections 4.3.1 and 4.3.2.

Figure 8 – Tournament selection example

Our tournament design uses pairs of individuals, selecting the best of both to enter the mating pool. As presented in Figure 8, the best individual survives the tournament, however, the first considered value in the comparison of two individuals is the one with lower infeasibility value. The cost of the solution represented by the individual (feasibility value) is only compared for individuals with the same infeasibility value. Our priority is to obtain feasible global solutions and only after we try to reduce the cost.

Variation operators

Considering that our solution representation has different content in each gene according to the locus of the gene, currently we have only considered the simpler crossovers, with one and two crossover points. In the one-point crossover, as exemplified in Figure 9 for a decomposition with five subproblems, a point between two genes is randomly selected and then each offspring receives a segment from each parent. The first receives the genes from the second parent between the start and the crossover point, and the remaining genes from the first parent, from the crossover point to the end. The second receives the same segments, but in the inverse order.

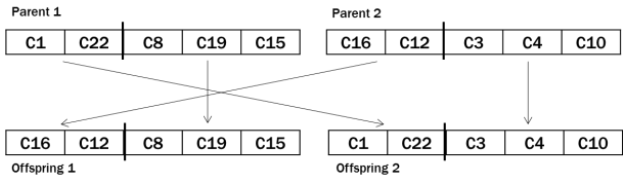

Figure 9 – One-point crossover

The two-point crossover, illustrated in Figure 10, is similar to the first, but selecting randomly two points where the crossover is done. With the two-point crossover, each offspring receives two segments of genes from one parent and one from the other. In the example presented in the figure, since

the central segment includes only one gene, each child receives only one gene from one of the parents and all the others from the other.

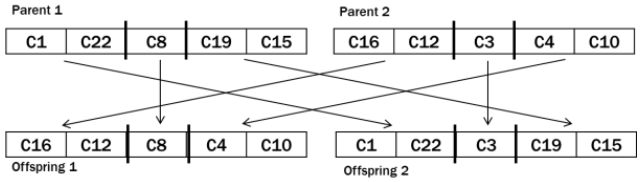

Figure 10 – Two-point crossover

The crossover operator allows the transfer of characteristics from both parents to the offspring. However, sometimes an individual has own characteristics that neither parent had. This phenomenon is designated as mutation and is also an operator implemented. This operator is considered a variation operator since it allows the search of a distinct zone of the search space. In the context of the SearchCol, it allows to obtain a distinct solution from the original pool of solutions containing all the subproblem solutions obtained during the CG stage. Its use is important to avoid being stuck in a local minimum when the other solutions, or parts of solutions, (sets of genes) are not included in the current population.

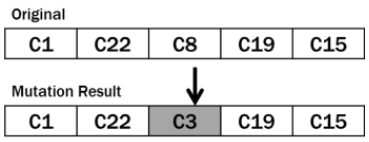

Figure 11 – Mutation example

The mutation operator is usually applied after the crossover operator to include in the offspring one or more new characteristics by changing the genes content. In our algorithm design, each gene has a low probability (defined by a parameter) to suffer a mutation but, if selected to mutate, a new subproblem solution is selected from the poll of feasible solutions to the subproblem associated to the locus of the mutated gene.

In the example presented in Figure 11, in the original chromosome the solution for the third subproblem is the one represented by column 8 in the RMP. The mutation operator was applied in the subproblem gene and the solution represented by column 3 was randomly selected from the pool of feasible solutions for that subproblem. In our algorithm, the mutation

operator allows the inclusion of new subproblem solutions in the population if the selected one was not part of any individual that compose the population.

4.2.3 Global Solution Repair Operator

Regardless of the metaheuristic used in a SearchCol based algorithm, the metaheuristic searches for the best combination of solutions stored in the pools during the column generation in order to respect the linking constraints and optimize the objective function value.

During the search phase, it can be useful to add new columns. However, after stopping the CG phase, the only option to generate new columns was using perturbations to fix some of the original variables and run again the CG. This process would imply stopping the metaheuristic search and start over again with the new search space updated with the inclusion of the new columns.

Considering an arbitrary global solution, its evaluation may show that some of the linking constraints are not respected. In some cases, this solution can be cleaned (when the linking constraints are not respect with a surplus) and completed (when the linking constraints are not respect with a positive slack). The sequence of cleaning and trying to complete an infeasible global solution constitutes the new repair operator proposed.

Even if the described repair procedure is simple, it is subject to some restrictions, namely:

- The procedure is dependent of the problem type;

- The existing subproblem solutions cannot be changed, since they were used to generate a column in the RMP and they can be part of other global solutions. If a subproblem solution would be changed, all the global solutions containing that subproblem solution will have the evaluation wrong if they were not re-evaluated.

- The subproblem constraints must be considered when changing a subproblem solution.

Considering the enumerated restrictions on the implementation of a repair operator for a global solution and using the knowledge about the SearchCol

framework implementation, some extensions were made in the framework to allow its implementation.

Since the repair of a global solution of a problem depends on the knowledge about the problem, a new operator was included in the framework. With this operator, each concrete decomposition can define its own repair procedures considering a starting solution, all the knowledge about the problem (particularly the constraints) and access the already existing solutions in the pools.

Independently of the problem type, and considering the restriction on the changes on the existing subproblem solutions, another change was introduced: if one of the subproblem solutions is changed, resulting in a new solution, the new solution is included in the pool of solutions of the corresponding subproblem and a new column is added to the RMP as occurs during the CG stage.

With this last feature of including new solutions for the subproblems, and the corresponding columns in the RMP, the proposed repair operator becomes an alternative to the use of perturbations (already available in the framework) to expand the search space from within the metaheuristic, avoiding stopping the current search and starting again a new one.

4.3 Evolutionary Algorithm based on Column Generation

The SearchCol concept proposes the use of a metaheuristic to execute the search phase after the conclusion of the column generation, as explained in Section 4.1. Besides the inclusion of a new model in the SearchCol framework, our main purpose is to use Evolutionary Algorithms (EA) as a new metaheuristic available in the framework. EA were proposed by (Holland, 1992) and its use in optimization is very frequent (Reeves, 1997).

In this section, details about the new EA integrating the SearchCol framework are presented.

4.3.1 Solutions Representation

Considering the origin of the EA, proposed by Holland (1992), that is the evolution of the biological species, each individual is completely described

by a chromosome allowing to identify all its characteristics by setting values in each gene. To implement an EA, we need to define the structure of the chromosome describing all the characteristics of an individual that represents a solution. The decisions to be made are the size of the chromosome (number of genes), which characteristic is defined by each gene (according to its location – locus) and which information is included in each gene (possible values).

In the context of the generality of the problems in the SearchCol framework, the objective is to find the best global solution. A global solution consists in a selection of a feasible solution (column) for each subproblem, in conformity with all the global constraints. When the column generation ends, we have, for each subproblem, a pool of feasible solutions which were generated for the corresponding subproblem during the column generation iterations. The definition of a global solution can then be simplified to the selection of a solution from each of these pools. Knowing that, we can define a chromosome to represent a SearchCol global solution with a gene to represent each subproblem solution. Since all these subproblems solutions are already stored, the gene needs only to contain a unique identifier of the subproblem solution, which is the identifier of the column generated with that solution.

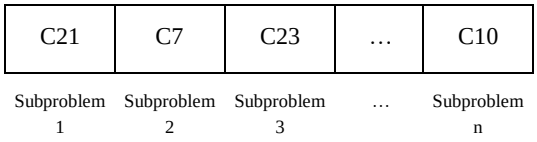

Figure 12 – Chromosome representing a global solution.

Figure 12 presents an example of a chromosome representing a global solution, where each gene position identifies a subproblem and the gene content identifies the solution/column selected to that subproblem. In this example, subproblem 1 has the solution associated to column 21, subproblem 2 the one from column 7 and so on.

4.3.2 Solutions Evaluation

One of the most important and sensible components of the EA implementation is the fitness function. The fitness function is the function used to evaluate an individual and is a core component for the EA operators,

particularly the selection operator, which uses it to distinguish the best in individuals' comparison.

In the SearchCol framework context, the evaluation of global solutions (individuals in the EA) considers two dimensions: the first one, called feasibility, is the cost of the solution, obtained by the objective function; the second, called infeasibility, is related to the number of linking constraints violated.

Since the evaluation depends on each specific problem, the evaluation is provided by the framework and can be tailored by each new problem if the general evaluation is not suitable. This function returns the evaluation values in the two dimensions, with priority on the infeasibility value, because the main objective is to find feasible global solutions (with all global constraints respected), and then the value of the solution (feasibility value) is considered.

4.3.3 Initial Population

The generation of a population of global solutions from the pools of subproblem solutions stored during the column generation was presented in Section 4.2.1. As occurs with the single solution metaheuristics that need an initial solution, the initial solution or population of those solutions is given as an input to the metaheuristic and so it is independent of the metaheuristic used. When setting the configuration of the global SearchCol algorithm, if a population-based metaheuristic is used, as is the EA, the population generator must be activated and parameterized with the single solution generators to be used as sources and the quantity of solutions included in the population from each source generator.

4.3.4 Selection and Variation Operators

The operators used in the EA are the ones already presented in Section 4.2.2. As selection operator the tournament selection is used and as variation operators, the one and two-point crossovers are available, as well as a mutation operator.

4.3.5 Elitism

The variation operators (crossover and mutation) are applied randomly in each generation to the individuals and it is possible that the resulting individuals are worse than the originals (parents or individual, before applying the operators). If this happens to the best individuals from a mating population, the next generation population will not include them. To prevent the loss of those best solutions through generations, the concept of elitism may be included to assure the persistence of the better solutions until the end of the algorithm.

The elitism consists in, in each iteration of the EA, select the best individuals from the population and moving them directly to the next generation. The number of individuals selected may vary starting from a minimum value of one, selecting only the best one. The increase of the number of individuals included in the elitism pool can originate a rapid stagnation of the population with identical individuals.

The positive contribution of the use of elitism strategies into the convergence of the genetic algorithms is known (Vasconcelos, Ramirez, Takahashi, & Saldanha, 2001). In (Deb, Pratap, Agarwal, & Meyarivan, 2002), the introduction of an elitism strategy was one of the improvements of the second version of the NSGA algorithm.

In our algorithm, we use a parameter to define the size of the elitism pool (parameter β) and when this parameter is greater than zero, the best β individuals are added to the elitism pool. Another parameter (ϕ) is used to define the percentage of individuals included in the mating pool by applying the selection over the elitism pool, to assure the presence of some of the best individuals as parents for the next generation. The resulting selection process is described in Figure 13. For each individual needed, the selection operator generates a pseudorandom number in the domain [0,1], which is used to define the origin pool from where the individual is picked, according to the probability that the number is higher or lower than the parameter ϕ.

The next generation population is filled with the offspring resulting from the crossover and mutation operators leaving the sufficient space to include the β best individuals from the previous generation.

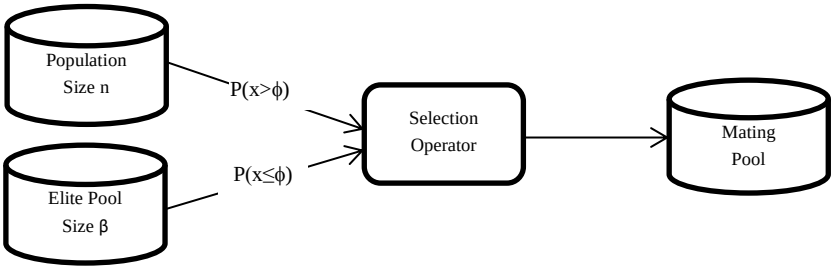

Figure 13 – Selection with elite population

4.3.6 Local Search

To try to improve the a selected solution (individual) in each iteration of the EA, a simple local search procedure was added to explore the neighbour solutions. Local search (Johnson et al., 1988) explores if neighbouring solutions are better than the current solution by testing small changes in solutions. The local search behaviour is similar to the mutation operator, but it does a more exhaustive search since it potentially tests the entire pool of subproblem solutions searching for an improvement.

Figure 14 shows an example of a global solution defined by the subproblem solutions {C1,C7,C23,C14,C10} and the pools of solutions obtained from each subproblem. Since each pool contains five solutions, the local search tests the replacement of the current one by each of the remaining. The possible replacements are illustrated assuming a single change and the test of all available solutions in the pools.

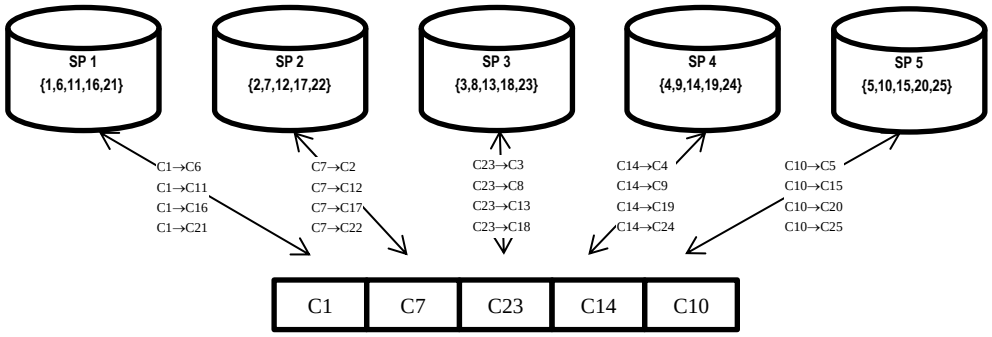

Figure 14 – Local search example

The local search metaheuristic implemented is configurable to stop when the first improvement is found in each subproblem pool or to test all available

solutions to each subproblem. It is also configurable to allow moves with only one change, two changes or the best between one and two changes.

In the EA, after applying the local search to an individual, if a better individual is found, it is added to the next population, increasing temporarily its size by one.

Apart from elitism and the local search, the usage of the repair operator introduced in Section 4.2.3 is also possible inside the proposed algorithm. The usage and its frequency are controlled by parameters used to configure the complete algorithm.

4.3.7 Complete Algorithm

The EA is described in Figure 15 by a flow diagram with the main activities and decisions.

Even if it is a preliminary activity, the generation of the initial population using the populations generator described in 4.3.3 is included in the diagram to help understanding. The second activity only occurs if the elitism is used (elite pool size > 0), moving the best individuals to a separate pool. Similarly to this one, all activities filled with the grey colour are only executed if using elitism.

In the "Selection" zone, tournaments are used to build the mating pool. The source of the individuals is again related to the configuration parameters.

In the "Variation" zone, the crossover and mutation operators are used to force the variation on the individuals to build the next generation by mixing the parents characteristics (crossover) and introducing small changes with new characteristics (mutation) distinct from parents. The probability of crossover and mutation depends on the configuration parameters.

The "repair" zone is where the repair operator is applied over each infeasible individual, if the application criteria is satisfied.

When the new population is full with the offspring, the individuals from the elite pool, if present, are added to the new population. After, if the local search is active, it is applied over the best individual of the new population and if a better solution is found, the new individual is added to the population.

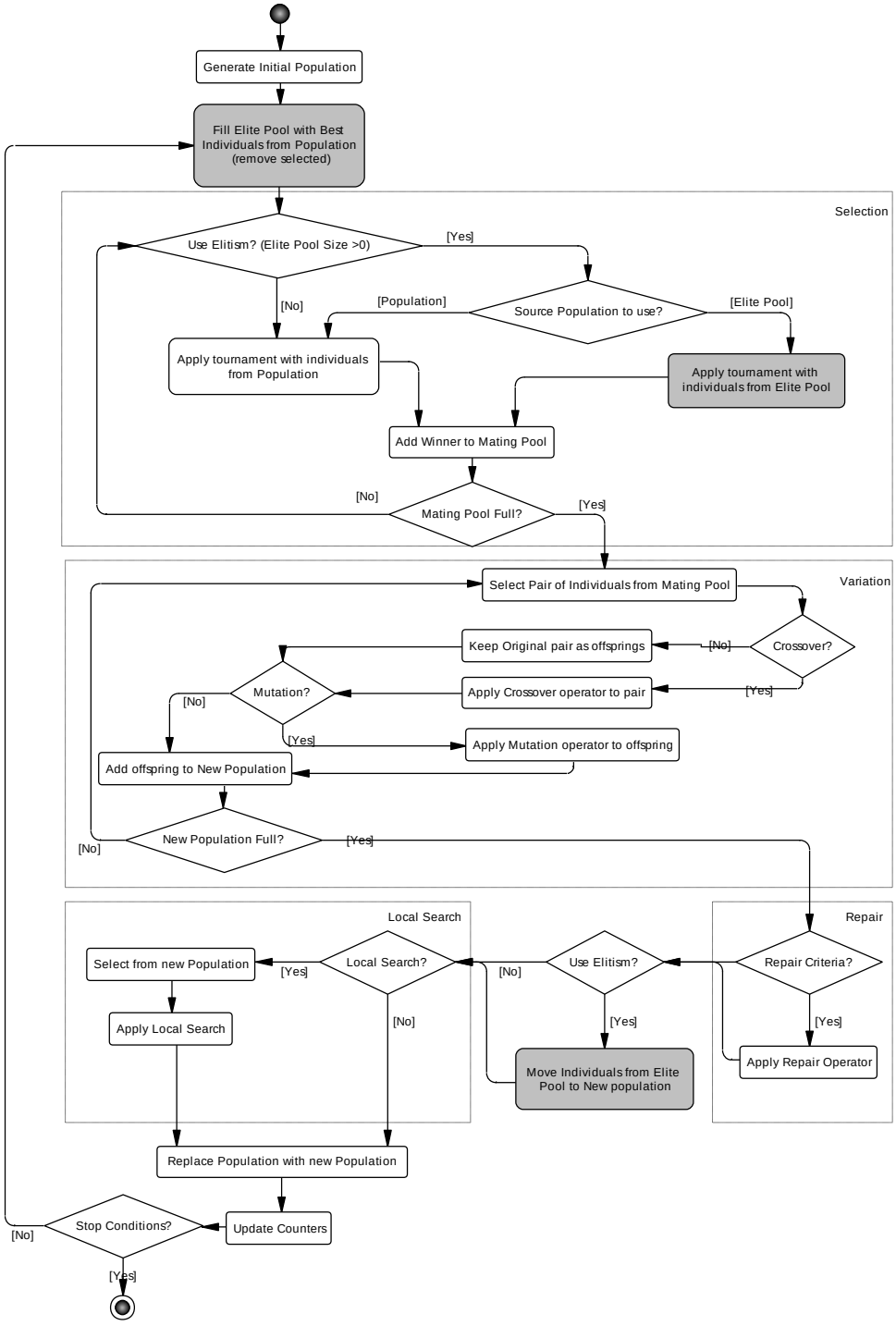

Figure 15 – Evolutionary algorithm flow diagram

The EA iteration ends with the replacement of the population by the new population and the update of all the counters used. The end of the algorithm

or the continuation to the next iteration depends on the test of the stopping criterion that is currently defined by a parameter that sets the maximum number of iterations without improvement in the best solution. The algorithm continues to the next iteration, repeating everything except the initial population generation, using the current population, updated in each iteration.

4.4 Summary

This chapter introduced the concept of search by column generation and presented the proposed extensions to the framework which implements the concept to allow using population-based metaheuristics to explore the search space. The way how the initial populations can be generated was explained followed by the details on the new operators proposed to be available for the population-based metaheuristics inside the framework. The global solution repair operator was also revealed as an extension to the framework, although it is not directed to any type of metaheuristic since it can be used by all of them.

The third section described the proposed evolutionary algorithm based on column generation, the solutions representation was explained as well as how each individual is evaluated. Besides the extensions described in the previous chapter, some additional features were proposed in the algorithm as are the use of elitism to prevent the loss of the best solutions and the possibility of using local search to try to improve solutions. The complete algorithm was described with all the components and configuration options.

5 Search by Column Generation for Bus Driver Rostering

SearchCol algorithms are developed according to the SearchCol framework (Alvelos et al., 2010), as described in Section 4.1, to address an optimization problem represented by a decomposition model using the combination of the column generation method, which is an exact method, with a metaheuristic, which is an approximation method.

In this chapter, the definition of a new decomposition in the framework is explained by describing the inclusion of the proposed BDRP decomposition. Details are given about the generation of the search space using the column generation, detailing the available configurations and its impact on the computational time and resulting search space. In addition, some of the enhancements implemented to improve the CG performance and change the resulting search space are also described. The chapter also describes the alternative metaheuristics available to explore the search space, besides the new evolutionary algorithm already introduced. Further, the concept of perturbations is introduced and a new perturbations generator is presented. The chapter ends with an overview of the algorithms used to address the BDRP in this research.

5.1 Implementation of the Models in the Framework

The models proposed in Chapter 2 were included in the SearchCol++ framework, an implementation in C++ of the theoretical SearchCol framework introduced in Section 4.1. The framework allows the inclusion of the compact model in order to obtain optimal solutions (when reached) by solving the problem directly with a branch-and-cut implementation provided by a commercial solver permitting the comparison of results of the SearchCol algorithms with optimal solutions (when possible).

A first class (*BDRostering*) stores the data from an instance of the problem. The internal data stores all parameters of the model and the class only needs

to implement the behaviour of a method responsible for reading the instance data from a file with a predefined structure. All the data and methods needed by the framework to load and solve the compact model are already defined in the framework. To include the BDRP compact model, a new class was defined inheriting from the class with the instance data (*BDRostering*) to access the model data and two methods (*LoadModel* and *LoadSolver*) were re-implemented. The first is responsible for loading all relevant information to represent the model and the second is responsible for loading the model into the optimization solver in use. After this, it is possible to solve the compact model.

The inclusion of the decomposition model in the SearchCol++ framework is similar: the framework defines all the internal data and all methods needed. Each specific decomposition creates additional internal data and overrides the methods where specific information from the particular problem is required. The class used to implement the BDRP decomposition model was named *DecBDRostering* (abbrev. of decomposition of bus driver rostering) and also inherits from the class *BDRostering* to access the problem data, similarly to the compact model implementation.

The two first methods to implement in a decomposition implementation are the ones responsible for reading the problem and loading the decomposition, *LoadInstance* and *LoadDecomposition*, respectively. The first loads the instance data according to the instances format defined for the problem. The second is responsible for building the decomposition model according to the data read from the instance. It is responsible for creating a new subproblem for each driver and defining inside each subproblem the constraints related to the represented driver, as defined in Model 6 or Model 7 or Model 8, and is also in charge for creating the restricted master problem, which includes making the linking constraints (*27*) and the convexity constraint (*28*) for each subproblem and also artificial variables to make the restricted master problem feasible at the start of column generation.

Concerning the optimization process, the new decomposition class also needs to implement two methods responsible for the subproblem optimization in each iteration of the column generation, the methods *SolveSP* and S*olveSPHeur*. The first has to solve the subproblem to

optimality using an implementation of an existing algorithm or call an external solver to optimize the subproblem (currently SearchCol++ uses the solver CPLEX (IBM, 2016b)) with the updated objective function coefficients. The second one allows the implementation of a heuristic to obtain feasible solutions to the subproblem (not optimal solutions) in each iteration, to speed up the column generation process if the heuristic is more efficient than the exact optimization.

The last two main methods implemented in the new decomposition class are the ones responsible for the subproblems objective function update and for the construction of a new column with a subproblem solution. The first, *IniModObj*, updates the objective function coefficients with the new dual variables values (π_{ih} and $\pi_{v)}$ from the last RMP optimization according to the function defined by (*33*) or the equivalent in the other subproblem formulations. The second, *SetColofSol*, receives a subproblem solution and creates a new column in the RMP, defining the coefficients for the convexity constraints (setting to one the coefficient of the constraint associated to the subproblem that generates the solution) and the correct coefficients to the linking constraints for each duty from each day assigned in subproblem solution (setting to one the coefficient of the constraint associated to the duty if assigned in the solution).

The previously introduced classes and methods reimplementation is the required code writing to define a new decomposition ready to be optimized, either by solving the compact model or use the CG to solve the decomposition model. Additional methods are implemented in the class representing the BDRP decomposition model, but they are related to the search stage and are introduced later.

The three models for the subproblem formulation were implemented inside the same class by using pre-processor directives for conditional compilation. Inside each of the methods implemented, the directives are used to define the code for each specific subproblem structure. The methods *LoadDecomposition* and *SetColofSol* have necessarily distinct code since the first builds the subproblems and the second converts the subproblem solution in a new column, and the structures of the subproblems and the solutions are distinct in the three models presented.

5.2 Operators for the BDRP decomposition

The new selection and variation operators included in the framework to support the population-based metaheuristics were described in the presentation of the evolutionary algorithm (Section 4.3) and do not need to be redefined for the BRDP since their behaviour is independent of the decomposition problem where they are used.

On the contrary, the evaluation of a global solution for the BDRP decomposition (a roster) can be customised to consider the characteristics of the problem. The same occurs with the repair operator which needs to consider the problem constraints in the assignment of new duties when trying to complete the roster.

The behaviour of the evaluation function of a roster, which is the evaluation of a global solution implemented in the BDRP decomposition, and the procedure used to repair an infeasible roster are detailed in the next sections.

5.2.1 Evaluation Function

The implementation of the evaluation function is simple and efficient, and it needs to be, since the fitness function is computed many times during the EA generations. To obtain the feasibility value of an individual, we only need to sum the cost of each individual schedule with the original costs, which is stored with the subproblem solution in the SearchCol shared data. To obtain the infeasibility value, a simple procedure is used because all the subproblems have the same number of variables representing all the duties shared (in the original definition (*13*), only the index *v* distinguishes the driver, inside the subproblems, the variables only have the indexes for the day number and the duty number). The procedure consists in overlapping all the subproblems solutions included in an individual (with ones in the duties assigned) and sum the values in the same position, obtaining a vector with a position to each duty from each day (as in the subproblem solution) and in each position the content is the number of times the duty was assigned. The number of unassigned duties is the count of positions with the value zero.

Figure 16 shows an example of the procedure to evaluate a global solution. Three drivers have feasible schedules with one duty assigned in each of the three days of rostering period considered in the example. The last vector

results from the sum of the three drivers' schedules, it results in the count of the number of assignments of each duty among all the drivers in the roster. This solution has an infeasibility value of 2, since the duties with index 1 from the first and the third day (1.1 and 3.1) were not assigned, having a zero value in the assignments vector. The example also includes the feasibility value. As previously explained, the cost of the global solution (roster) is the sum of the costs of all the drivers/subproblems' solutions.

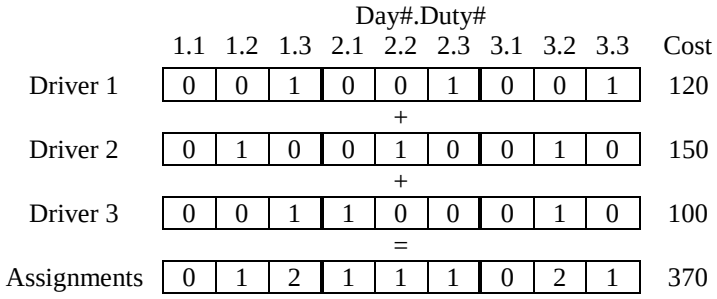

Figure 16 – Evaluation procedure

For the BDRP we define the infeasibility value considering the count of over-assigned duties (neglecting the first assignment of each duty, which is expected), the number of under-assigned duties (number of duties not assigned in the roster) and the number of additional days-off counted in the roster. To obtain the "normal" number of days-off in a roster we consider that each driver can have $days$-$(weeks \times d_w)$ duties assigned and so the minimum number of drivers used in the roster is (the parameters were introduced in Section 3.2):

$$min\_drivers = \lceil total\ number\ of\ duties/(days - (weeks \times d_w)) \rceil. \quad (74)$$

The total number of days-off expected in the roster is $min\_drivers \times days - total\ number\ of\ duties$, which corresponds to the obligatory days-off of each driver plus the remaining days-off of an eventual driver with incomplete schedule.

The final value of infeasibility of a global solution (roster) results from:

$$\text{Infeasibility} = \begin{pmatrix} inf_{c1} \times under\_assigned\_count + \\ inf_{c2} \times over\_assigned\_count + \\ inf_{c3} \times extra\_days\_off\_count \end{pmatrix} \quad (75)$$

Where coefficients $inf_{c1}$to inf_{c3} are used to define the contribution of each factor to the final value.

5.2.2 Roster Repair Operator

The overview of the repair operator behaviour and its objectives was introduced before. In this section, the particular implementation of the repair operator to global solutions of the BDRP is described.

Considering a random global solution, independently of the cost of the solution, it needs to assign all the duties in order be considered feasible, but the over-assignment can also occur, as was the case in the example from Figure 16 where two duties were never assigned (duties 1.1 and 3.1) and two other were assigned twice (duties 1.3 and 3.2). The over-assignment in a solution has two consequences: the first is the cost associated to the worktime of the duty and the second is that when a driver has a duplicated duty assigned to him in a day, it avoids the assignment of a duty from that day that is not assigned yet, resulting in the need for more drivers and an ineffective occupation percentage of the drivers.

To solve the problem observed, neglecting the subproblem constraints, one solution could be the replacement of a schedule removing the repeated duties and inserting the missing ones, as illustrated in Figure 17. The old work-schedule for driver 3, highlighted in grey, is replaced by a new one where the duties assigned to that driver on the day 1 and 3 are changed to the ones that were not assigned, resulting in a roster without under and over-assignment.

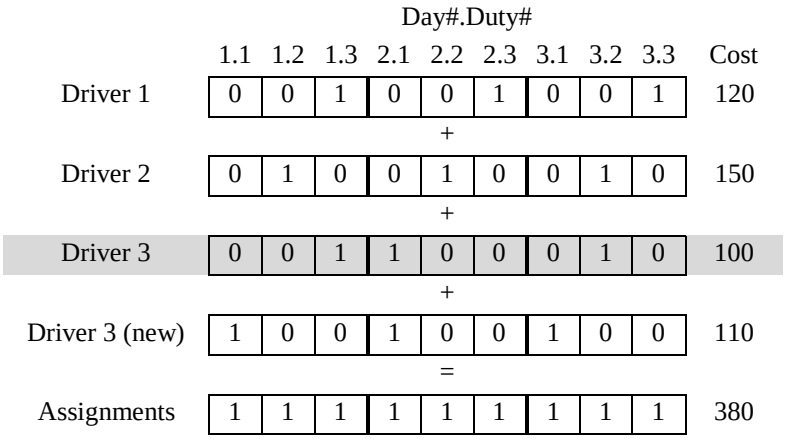

Figure 17 – Repair procedure example

In general, overcoming under-assignments involves a more elaborated procedure discussed below, after the implementation description.

Figure 18 describes the major interactions between the classes in the repair procedure. It is easily observed that all the repairing work is done in the decomposition implementation (Decomposition), represented by the methods *Clean* and *AssignMissingDuties* to denote the removal of the over-assigned duties and the try to assign the remaining ones, respectively. After the repairing, the new subproblem solutions are also saved inside the decomposition implementation, inside the object corresponding to the subproblem to which the solution belongs, making it invisible to the outside if new columns were generated or not. From the context of the EA (Metaheuristic), the *repair* method only asks the decomposition to repair a global solution and receives a new one repaired, which can be included in the population since all the components (subproblem solutions) are feasible and it was already evaluated to update the feasibility and infeasibility values. If new solutions were created during the process, they are stored in the pools of solutions (*SaveSolution*) and the corresponding columns are added to the RMP (*AddColumn*).

Figure 18 – Global solution repair sequence diagram

The cleaning stage of the roster repair operator has no constraints since we can always remove duties from a driver's work-schedule preserving its feasibility. However, to assign missing duties to incomplete work-schedules, the subproblem constraints must be checked in every assignment.

The assignment procedure used by the repair operator implemented for the BDRP decomposition is based in the roster heuristic (described in the next section). For each duty not assigned, its assignment is tested in all available drivers until the assignment succeeds or all drivers were tested.

After the initial assignment procedure, if there are still duties to assign, it is because the constraints do not allow the assignment, starting a second stage where, in each driver, the replacement of the currently assigned duty by the unassigned duty is tested, but only if the duration of the new duty is larger than the previous one (because the smaller duties will be easier to reassign). In this stage, the replacement is tested starting from the drivers with lower overtime cost, trying to minimize the cost by assigning the larger duties.

A third stage intents to test again the assignment of the remaining duties in the drivers with a day-off on the day of the duty to be assigned. In this stage, the idea is to move the day-off in the driver schedule (to the days before or the days after) and assign the duty. If the assignment succeeds, the duty replaced by the day-off needs to be re-assigned, but the procedure starts from the beginning. The move of the day-off is tested in the six neighbour days (three before and three after), testing always the "move before" and "move after" in each distance (1, 2 and 3 days).

When all the duties are tested, the new roster is saved by storing the resulting work-schedules (subproblem solutions) into the corresponding subproblem solutions pool and generating the corresponding column, whenever the subproblem solution is new.

The repair operator was included in the EA as an option to repair the individuals of the population (rosters) that have an infeasibility value greater than zero. Currently the algorithm can be configured by setting the interval (in iterations) between each usage, since it is computationally expensive and can make a significant change on the population. Figure 19 describes how the new repair operator can be included in any metaheuristic. The metaheuristic only needs to define the criteria to identify if a global solution

should be repaired and then apply the repair procedure in the solution to repair. As return, the metaheuristic receives the repaired solution and can continue the search with the new solution as a new point to evaluate.

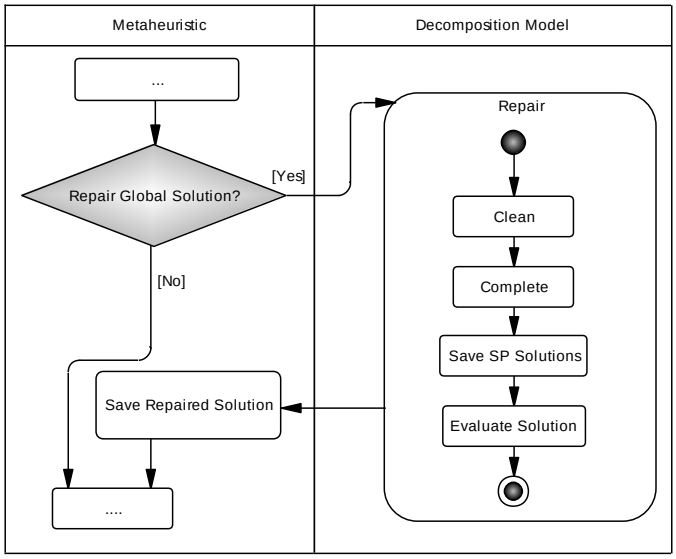

Figure 19 – Use of the repair procedure from the metaheuristic

5.3 Generation of the Search Space

The search space available to be explored by the metaheuristics results from the subproblems' solutions obtained during the column generation execution. The size of the resulting search space is highly dependent on the number of iterations the column generation needs to achieve the optimal solution and also on the number of subproblems. However, the framework allows to use different configurations on the column generation which impact the performance and/or the resulting search space.

When using the general CG a *tailing-off effect* is commonly observed. It consists in a slow approximation to the optimal solution (Lübbecke & Desrosiers, 2005). If a high number of iterations are expected, one approach to reduce the global computational time is by reducing the time in each iteration. One option is a deviation of the normal cycle, by changing the number of subproblems solved in each iteration or deciding if all columns are added to the RMP or only the best ones. In the framework presented in (Alvelos et al., 2013) these configurations are allowed when running the CG algorithm.

5.3.1 Column Generation Cycle Configurations

In the standard column generation method, in each iteration, all the subproblems are optimized and the corresponding solutions are tested as attractive to generate a new column in the RMP. Considering the decomposition model presented in Section 3.3, the number of subproblems is equal to the number of drivers, represented by *v*, which results in, in each iteration, spend on average v^*t_{SP} units of time, where t_{SP} is the average time used to optimize one subproblem and also add at maximum *v* new columns to the RMP and the corresponding solution to the search space, if all the columns are considered attractive.

The inclusion of all the attractive columns in each iteration results in a wider search space, which may be advantageous or worse to the search, but it penalises the column generation performance since a large number of variables is achieved in the RMP, increasing the time needed to its optimization through the iterations as the number of variables increases.

One of the configurations available allows to define that only one column is generated in each iteration, which reduces de growth of the number of variables in the RMP and consequently its optimization time, however the search space will also include less subproblem solutions.

In this second configuration, the subproblems continue to be optimized in each of the iterations of the column generation, however, only the one with the lowest reduced cost is added to the RMP and the corresponding solution added to the search space. This configuration is useful if detected that the RMP optimization consumes a high proportion of the total optimization time but it also needs additional iterations if the not added columns are needed for the optimal solution and are included in later iterations.

In the configuration where only the subproblem solution with the lowest reduced cost is used to generate a column, the eventual improvement in the performance results from the decrease on the time used in the optimization of the RMP through the iterations. Since all the subproblems continue to be optimized in each of the iterations, no reduction is expected in the total time used in the optimization of the subproblems. If the number of iterations needed to achieve the optimal solution do not grow until the RMP

optimization time reduction is reversed, the total performance is improved and the resulting search space will contain a lower number of solutions, when comparing with the first configuration.

In the previous configuration the eventual improvement in the performance results from avoiding the growth of the number of variables in the RMP, but no changes were made in the performance of the subproblems optimization.

A third configuration can be used where the time used in each iteration to solve the subproblems is also reduced and the maximum number of columns added to the RMP continues to be one. The reduction on the time is in the proportion of $1/v$ since only one subproblem is solved in each iteration. In this configuration, the selection of the subproblem to solve in each iteration can be made in two ways: in the first, the same subproblem is continuously solved (with updated objective function coefficient values) until it does not return attractive solutions to produce new columns and then the process continues with the next subproblem with the same behaviour; in the second, the subproblems are solved sequentially. If the first subproblem is solved, the corresponding solution is tested to generate a new column, and in the next iteration, after the RMP optimization and the update of the objective functions, the second subproblem is solved, continuing this cycle until none of the subproblems returns an attractive column.

The behaviour of the column generation cycle in the third configuration is described in the algorithm presented in Figure 20. If the subproblems are solved sequentially, the active subproblem is changed by the instruction on the line 11 of the algorithm, otherwise, if the same subproblem is continuously solved while returning new solutions, the active subproblem is only changed when the current does not return a solution allowing to generate a new column.

This configuration may result in a global improvement of the performance since it controls the growth of the number of variables in the RMP, adding a single column by iteration, and also reduces the time used to solve the subproblems by solving only one by iteration. The improvement achieved by this configuration is lost if the number of iterations needed to achieve the optimal solution, and the corresponding computational time, grows in the same proportion.

```
1: SP = first subproblem
2: REPEAT
3:     Optimize RMP
4:     Success = false
5:     DO
6:         Update SP objective function costs
7:         Solve SP
8:         IF (SP solution is an attractive column) THEN
9:             Add column to RMP with SP solution; Success = true
10:        IF SequentialMode THEN
11:            SP = next subproblem
12:        ELSEIF not Success THEN
13:            SP = next subproblem
14:    WHILE (not Success and SP ≤ numberOfSubproblems)
15: UNTIL CG stopping criterion fulfilled
```

Figure 20 – Column generation cycle with single subproblem optimization algorithm

5.3.2 Symmetry Breaking Constraints on the Subproblem Model

One of the difficulties observed when addressing staff rostering problems is the existence of symmetrical solutions (Walsh, 2006; Yunes et al., 2005) which hinders the optimization by the need to explore a larger search space where there are no better solutions.

In (Walsh, 2006), an example of existence of symmetry in staff rostering problems is the exchange of the work-schedule between two workers. If the two workers do not have particular constraints for their schedule definition and have the same cost, the swap results in an equivalent solution. The same can occur in the BDRP we are addressing, but the selection of the work-schedules for each driver is made in the RMP by combining multiple work-schedules.

Inside the definition of the work-schedule, the symmetry can also be observed. In (Law, Lee, Walsh, & Yip, 2007), two types of symmetry are described: interchangeable variables and values. The first comprises the solutions that are equivalent if two variables are interchanged. In the definition of a work-schedule, into the subproblem of the BDRP decomposition model, the same can occur.

The selection of the duties is restricted by the feasibility of the sequence, respecting the rest time and intervals between days-off, the total work-time

for the rostering period and the total work-time for each week. Even with these constraints, there are two types of interchanges on the assigned duties that do not change the cost and feasibility of the solutions. In the data instances used for testing the BDRP model (Moz et al., 2009) the same set of duties is used for the week-days and another for the weekend days, and so, if for the days h_1 and h_2 the duties assigned are the ones with index x and y, if the interchange of the duties between the two days is still a viable sequence, the resulting work schedule is feasible and has the same cost. The possibility of replacement of a duty in a work-schedule without changing its cost can be seen as the interchangeable variables described in (Law et al., 2007).

As suggested by Walsh (2006) for the general staff rostering problems, in the case of the instances we use for the BDRP, for some subsets of drivers, the work-schedules can the interchanged between pairs of drivers. The drivers can be grouped in the same subset if they have the same cost for each overtime time unit and have the same constraints for the definition of the type of duty on day zero (depends on the duty assigned on the last day of the previous roster).

The identification and elimination of all the symmetric solutions can be intractable or considered very hard (Walsh, 2012). Inside the subproblems defined by Model 6, we cannot easily include constraints to eliminate symmetries, particularly because the symmetric solutions need to be evaluated considering the complete problem (all the subproblems/drivers), however, considering the existence of groups of drivers with similar characteristics previously described, a set of constraints can be added in each of those groups to restrict the assignment of duties in the first day of the rostering period.

Considering each group of drivers from the subset with identical characteristics g_v and the set of duties for day zero T_0^w ordered by decreasing size on the duration of the duty stored in vector F, we can define that if one of the drivers has the first duty assigned, that driver is the first from the group, and the same for the second duty and following duties until the size of the group of drivers. If this is not the case, the first duty has to be assigned to some driver of other group. In practice, the first duty can only be

assigned to the first driver in the group, the second can be assigned to the first or the second, the third can be assigned to the first, second or third driver, etc.

Formally, in the subproblem of driver with index *v* in each group, the following constraint is added:

$$\sum_{i=1}^{v-1} y_{F[i]1} \leq 0$$

Where $y_{i1}$is the binary variable from the subproblem representing if duty *i* on day 1 is assigned to driver *v* (represented by the subproblem) or not and $F[i]$ is the original index of the duty from the vector *F* with the ordered duties from T_0^w.

The first constraint is included in the second subproblem of the group (*v*=2), for avoiding the assignment of the first duty, which can only be assigned to the first subproblem. The number of duties which the assignment is conditioned depends on the size of the groups. For each group of size *s*, *s*-1 duties will be restricted in the last driver of the group.

Even if only some of the variables related to the first day of the rostering period are considered, the new constraint was included as an option in the creation of the subproblems in order to try to reduce the computational time used in the optimization with the exact solver.

5.3.3 Heuristic Solutions for the Subproblems

Besides changing the normal path of the CG as presented in Section 5.3.1, a usual approach is to use efficient combinatorial algorithms or heuristics to solve the subproblems, if available, reducing considerably the optimization time. Multiple examples are found in the literature where dynamic programming (Cintra & Wakabayashi, 2004), constraint programming (Yunes et al., 2005) and heuristics (dos Santos & Mateus, 2009) are used to obtain subproblem solutions.

Considering the computational time observed with the classical CG in the optimization of the decomposition model for the BDRP and since multiple configurations of the CG algorithm path are already available in the framework where the algorithm is being implemented, a heuristic to obtain

feasible solutions for the subproblems was developed (therefore, avoiding the use of the exact solver).

Dynamic programming was also considered to solve the subproblems, however, even if the stages are easily defined by the days in the rostering period, the same does not occur for the definition of the states. In each stage the decision to take is to select a duty for a driver, but the decision depends not only in the previous duty selected but also needs information about the accumulated work-time (in the week and total), the number of days since the last day-off and if a day-off was already assigned on a Sunday or not. The space of states grows exponentially along the stages since each one is defined by all the decisions taken in previous stages and dynamic programming becomes similar to complete enumeration.

The heuristic used to solve the subproblems of the BDRP decomposition model is based in the decoder algorithm proposed in (Moz et al., 2009). The objective of the heuristic is to build schedules with the highest contribution to improve the global solution by testing the assignment of the duties previously ordered.

Independently of the order by which a duty is selected to be assigned to a driver, in the definition of the work-schedule, the constraints of the subproblem prevent some of the assignments. All the situations where the assignment of the duty with index *i* on day *h* is not allowed are illustrated and described in the following paragraphs.

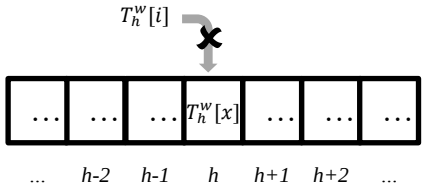

Figure 21 – Invalid assignment: day with duty already assigned.

The day of duty *i* has already a duty assigned: each driver has only one duty assigned on each day. If the duty with index *x* on day *h* was already included in the work-schedule of the driver, as shown in Figure 21, the current duty with index *i* from the same day (*h*) cannot be assigned.

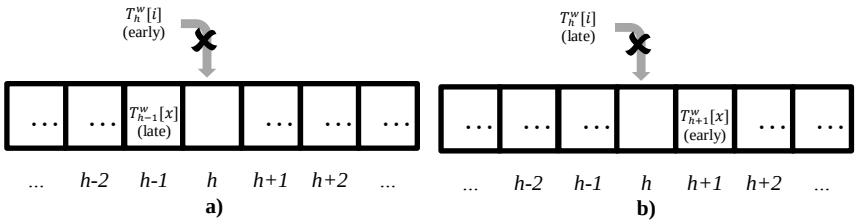

Figure 22 – Invalid assignment: insufficient rest time between duties

Duty *i* is incompatible with the assigned duty on (day of duty *i*)-1 or (day of duty *i*)+1 considering the minimum rest time between duties. Duties are classified as "early" or "late". An early duty cannot be assigned in the next day after a late duty, as shows Figure 22 a), a late duty cannot be assigned in the previous day of an early duty, as shows Figure 22 b).

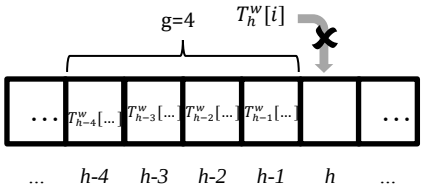

Figure 23 – Invalid assignment: maximum number of consecutive work-days.

The assignment of duty *i* makes a sequence of working days (without a day-off) longer than the maximum allowed. The assignment of the duty on day *h*, as shows the Figure 23, results in five consecutive days without a day-off, exceeding the maximum defined by the parameter *g*, with the value 4 in the example. In this case, the day *h* must have a day-off assigned.

The assignment of duty *i* exceeds the maximum of working time units allowed by week or for all the rostering period. Figure 24 a) represents the situation where the total time of the duties assigned in the week (from which the day *h* belongs) only has a slack for a duty with duration of *y* units of time and the duration of duty *i* is greater than *y*, preventing the assignment of the duty to respect the week total worktime limit defined by parameter b_1. Figure 24 b) represents the situation where the total time of the duties assigned in the complete rostering period only has a slack for a duty with duration of *x* units of time and the duration of duty *i* is greater than *x*, preventing the assignment of the duty to respect the total worktime limit for the rostering period, defined by parameter b_2.

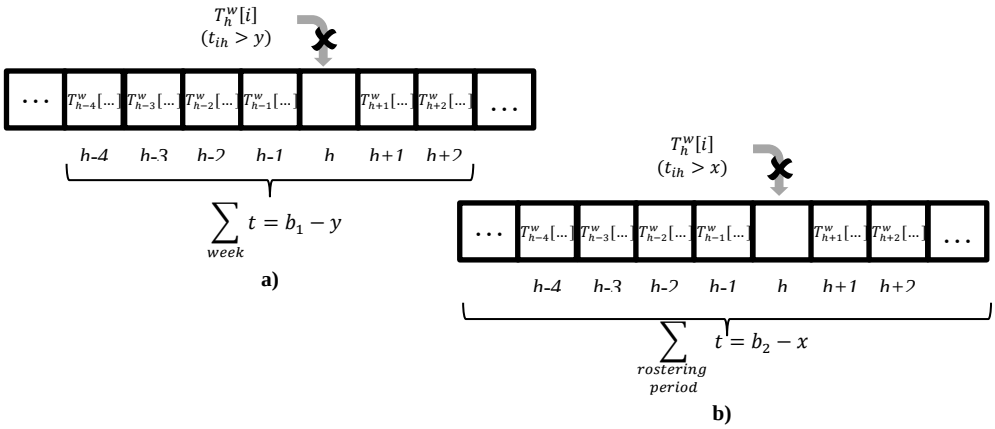

Figure 24 – Invalid assignment: Maximum number of worktime units by week/rostering period.

Assignment of duty *i* makes impossible to ensure the minimum number of days-off in each week of the rostering period or the minimum number of days-off on Sundays in all the rostering period. Figure 25 a) shows a situation where the assignment of the duty on day *h* is not allowed because the count of days-off in the week of the day *h* (considering the day *h* as free) has reached the minimum, defined by the parameter d_w. Figure 25 b) shows a situation where the assignment of the duty on day *h*, assuming the weekday of *h* is a Sunday, is not allowed because the count of days-off on Sundays in the complete rostering period (considering the day *h* as free) has reached the minimum, defined by the parameter d_s.

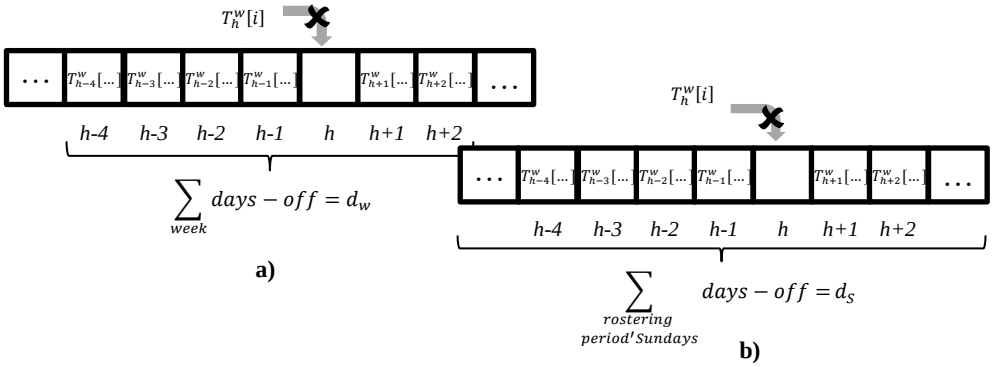

Figure 25 – Invalid assignment: Minimum number of days-off by week/on Sundays.

To build a work-schedule (subproblem solution) using the heuristic, in each iteration of the CG, the duties are ordered in ascending order according to the updated costs and then it starts with an empty work-schedule and picks

the next duty to assign according to that order. The assignment of each duty in the work-schedule is tested and if one of the previous situations is identified, the assignment fails, and the next duty is selected, otherwise the assignment succeeds and the work-schedule is updated and then the next duty is selected for assignment. When all the duties were tested, the work-schedule is filled with days-off in the days without duties assigned.

The heuristic algorithm to build a driver schedule is described in Figure 26. It builds a schedule for a driver trying to assign the duties with the most negative costs (after the update of the objective function with the dual solution of the RMP) following a greedy behavior. The function *TestAssignment* used in the heuristic algorithm tests all the conditions previously enumerated, which represent the constraints of the subproblems formulation. If any of the conditions fails, the function returns *false* and only if all the conditions are verified the function returns *true*, allowing the assignment of the duty to the schedule of the driver represented by the subproblem.

```
Get dual solution from RMP optimization (π);
Update objective function of the subproblem;
Order updated costs (costs[]) in increasing order, keeping information from original
                    position of duty i (origDuty[i]); -
Build empty schedule for the rostering period size;
Initialize driver data: working time (total and week);
FOR i=1 to size of costs[]
    IF costs[i]>0 THEN
        Next i;
    Assign=TestAssignment(origDuty[i]);
    IF Assign THEN
        set driver as full in the day of origDuty[i];
        Update schedule: add original cost of origDuty[i] to schedule;
        Update driver data: add origDuty[i] time length to total working time and
                                corresponding week working time;
FOR d=1 to number of days of the rostering period;
    IF no duty was assigned to driver on day d THEN
        Assign a day-off to driver on day d;
IF number assigned duties >0 THEN
    Update schedule: add fixed cost of driver use;
Return schedule;
```

Figure 26 – Driver schedule builder heuristic algorithm

Having a heuristic to obtain solutions to the subproblems, the column generation algorithm is changed to use the heuristic, since it does not replace the exact optimization solver, because the solutions of the heuristic

are not optimal, only feasible. The resulting algorithm is presented in Figure 27 and details the column generation using the heuristic.

```
DO
    Optimize RMP;
    Update subproblems objective function with current dual solution of the RMP;
    FOR EACH subproblem
        Solve using heuristic;
    Add new columns into the RMP with subproblems attractive solutions;
    IF no new columns added THEN
        FOR EACH subproblem
            Solve using exact optimization solver;
        Add new columns into the RMP with subproblems attractive solutions;
WHILE new columns added >0
```

Figure 27 – Column generation with subproblem heuristic algorithm

In the new configuration of the column generation cycle, the heuristic is used until no new columns are added from the obtained solutions. At that point, the exact optimization solver is used to obtain the optimal solutions of the subproblems and eventually add new attractive columns. In the next iteration the heuristic is tested again.

In the SearchCol++ framework, the algorithm presented in Figure 27 can have other configurations. It is possible to solve only a single subproblem in each iteration, optimize the RMP again and, in the following iteration solve the next subproblem, iterating by all the subproblems. This strategy results in less columns added to the RMP when the subproblems are returning similar solutions, allowing a faster optimization of the RMP, due to a reduced number of variables.

5.3.4 New Rosters using Column Generation

To assure the existence of complementary schedules between each other, we now present a heuristic that, in the column generation cycle, assigns the duties considering all the subproblems together, as a single one. The cycle does not change, however, instead of generating individual schedules one by one, a new heuristic is used to generate a feasible combination of schedules as well as the schedules *per se*. The primary purpose of solving the subproblems in an aggregated way was to assure the existence of complete or partial rosters without the over-assignment of duties whenever the column generation was stopped. If the solutions included in the initial

population of the evolutionary algorithm are already feasible rosters, the expected result of the evolution is a better roster.

The heuristic presented in Figure 28 is able to build rosters by testing the assignment of each of the available duties in the schedules of free drivers. Since in each iteration of the column generation a new dual solution is used to update the costs of the duties in the subproblems, the order in which the duties are assigned may vary from iteration to iteration. The objective is that the dual solution of the RMP can guide the generation of distinct, and feasible, rosters through the iterations.

When using the aggregated heuristic in the column generation algorithm in Figure 27, the cycle solving the subproblems is replaced by a single call to the new heuristic, which returns schedules for all subproblems/drivers. The exact solver continues to be used when no new attractive columns are built from the heuristic solutions.

The BDRP model defines a cost to each unit of time of overtime which may be different to between drivers. However, in our test instances, the drivers are split in a limited number of categories. All the drivers in the same category have the same cost for the overtime labor. This means that we still want to assign first the duties with bigger overtime to the drivers from the category with lower cost of overtime, if possible. However, we want to distribute them among all, avoiding the schedules with extra days-off because of a large concentration of duties with overtime.

Although the ability of the Roster Builder Heuristic to generate feasible and distinct rosters, preliminary tests showed that the schedules of the first drivers were filled with the duties with higher overtime. Even if we want to assign the duties with higher overtime to drivers with lower salary, which are the first group in the set of all drivers, if the assignment starts always from the same driver, his/her schedule will be filled with the duties with larger overtime, resulting in an unbalanced work distribution.

Given the existence of different drivers' categories, concerning the value paid by overtime labor, drivers of the same category are grouped, and the dual solution values of the convexity constraints are used to order them inside each group.

Get dual solution from RMP optimization (π);
Order duties (*duties[]*) in ascending order of the dual solution value of the linking constraints, keeping information from original position of duty *i* (*origDuty[i]*);
Order drivers (*drivers[]*) in ascending order of the dual solution value of the convexity constraints;
Build an empty schedule for the rostering period size to each of the available drivers (subproblems);
Initialize drivers data: working time (total and week);
FOR *i*=1 to size of *duties[]*
    FOR *v*=1 to *number of drivers*
        Select schedule of *driver[v]*
        Assign=**TestAssignment**(*origDuty[i], schedule [driver[v]]*);
        IF Assign **THEN**
            Set driver *v* as full in the day of *origDuty[i]*;
            Update schedule: add original cost of *origDuty[i]* to *driver[v]* schedule cost;
            Update driver *v* data: add *origDuty[i]* time length to total working time and corresponding week;
            EXIT FOR
FOR *v*=1 to *number of drivers*
    FOR *d*=1 to *number of days of the rostering period*;
        IF no duty was assigned to driver *v* on day *d* **THEN**
            Assign a day-off to driver *v* on day *d*;
    IF number assigned duties to driver *v* >0 **THEN**
        Update driver *v* schedule: add fixed cost of driver use;
Return *schedule[]*;

Figure 28 – Roster builder heuristic algorithm

To assure that when the dual values of the convexity constraints do not lead to the desired diversity in the order of the driver inside each group, we added an additional procedure to select the first driver inside each ordered group. We started considering each group of drivers as a circular array. After that, two configurations were prepared to define how a driver is selected when a new duty needs to be assigned.

By default, when a new duty is selected for assignment, the driver to select is the one in the position 1 of the first group. We developed two configurations of the Roster Builder Heuristic with drivers' rotation, namely the sequential and the random configurations. In both, after the assignment of a duty, we rotate the drivers inside the group, the first is removed and inserted at the end. In the sequential configuration, the rotation is of a single position, and in the random configuration, the number of positions rotated is randomly selected between one and the number of drivers in the group minus one, to avoid a complete rotation to the same position.

The inclusion of the rotation leads to a better distribution of the duties with overtime among the group drivers. Figure 29 presents the algorithm of the

roster builder heuristic with drivers' rotation. The changes are: the inclusion of the groups of drivers, the selection of the configuration: 'normal' – without rotations; 'sequential' – to rotate one position, picking the drivers sequentially; 'random' - using the stochastic selection by rotating the driver inside the group using a random number of positions.

The schedules composing the roster are saved in the poll of solutions whenever considered attractive by column generation.

```
Get dual solution from RMP optimization (π);
Order duties (duties[]) in ascending order of the dual solution value of the linking
    constraints, keeping information from original position of duty i (origDuty[i]);
Split drivers in groups with the same category of salary;
Order drivers inside each group according to the dual solution value of the corresponding
    convexity constraint;
Build an empty schedule for each of the available drivers (subproblems);
Initialize drivers data: working time (total and week);
FOR i=1 to size of duties []
    FOR g=1 to size of groups of drivers
        Select starting driver position according to configuration r= (0 or 1 or random);
        FOR j=1 to r
            Rotate drivers inside group (remove from the beginning and add to the end);
        FOR v=1 to number of drivers in group g
            Select schedule of driver v
            Assign=TestAssignment(origDuty[i], schedule[v]);
            IF Assign THEN
                Set driver v as full in the day of origDuty[i];
                Update schedule: add original cost of origDuty[i] to driver[v]' schedule;
                Update driver v data: add origDuty[i] time length to total working time and
                                      corresponding week;
                EXIT FOR
        IF Assign THEN EXIT FOR
FOR v=1 to number of drivers
    FOR d=1 to number of days of the rostering period;
        IF no duty was assigned to driver v on day d THEN
            Assign a day-off to driver v on day d;
    IF number assigned duties to driver v >0 THEN
        Update driver v schedule: add fixed cost of driver use;
Return schedule[];
```

Figure 29 – Roster builder heuristic with drivers' rotation algorithm

5.4 Single-solution Metaheuristics

The stage which follows the column generation is the search for an integer solution combining the solutions from each subproblem. To find the best combination of those solutions, namely, explore the search space composed by all the subproblem solutions obtained during the column generation, the SearchCol framework suggests the use of metaheuristics.

Besides the new population-based metaheuristic proposed, some single solution metaheuristics are already available in the framework implementation. In the next sections, some of the most relevant are introduced since they were useful in the development of this research. The MIPSearch (see Section 5.4.1) was used to evaluate the search spaces resulting from the multiple column generation configurations, the Local Search was integrated in the proposed evolutionary algorithm to improve the global solutions and the Variable Neighbourhood Search and the Simulated Annealing metaheuristics were used to assess the ability of a preliminary version of the new metaheuristic (evolutionary algorithm) in exploring the search space to find good quality integer solutions.

5.4.1 MIPSearch

The MIPSearch consists in transforming the RMP into a MIP by setting the variables related with the generated columns to integer variables and solve the resulting problem using a commercial implementation of the branch-and-cut exact algorithm (IBM, 2016b).

In fact, as branch-and-cut is able to achieve the optimum solution, if it exists using the variables included in the RMP and necessary time is given. If the number of generated columns is large, the method may face the same computational difficulties to achieve the optimum solution as occur in the original global problem.

The MIPSearch can be used as a heuristic by setting a time limit in which the commercial solver, using improved preprocessing strategies are able to achieve good solutions in an initial stage of the optimization.

5.4.2 Local Search

A local search metaheuristic (Aarts & Lenstra, 1997) was already introduced in Section 4.3.6 as an improvement feature of our evolutionary algorithm.

In the context of the SearchCol algorithms, the local search algorithm starts with a global solution. Then, for each of the subproblems, the algorithm tests the replacement of the currently selected subproblem solution by the others inside the pool of solutions of the same subproblem. The algorithm

can be configured to stop when the first improvement is found for each subproblem or to test all possible swaps.

5.4.3 Simulated Annealing

The simulated annealing (SA) metaheuristic was proposed in (Kirkpatrick et al., 1983) and is an optimization method based in the metal cooling physical process (the annealing process) which prevents the solutions from getting stuck in local optimums by allowing not only improvements but also moves to worst solutions in a low, but dynamic, probability of occurrences. A good description of SA can be found in (Dowsland, 1993).

The distinct characteristic of the SA metaheuristic is that it allows to jump to a worse solution, which is a mechanism to avoid the local optimums. The algorithm uses a parameter designated as temperature which is used to control the probability of accepting a worse solution. Generally, the algorithm includes a cycle to iterate the distinct temperature levels (updated by a factor given by a parameter) between the initial and the minimum temperature. In each temperature, an inner cycle evaluates random neighbor solutions. In the SearchCol, the number of neighbor solutions evaluated is defined by a parameter which is multiplied by the number of subproblems. The acceptance probability is calculated by $\alpha = e^{\frac{\Delta s,s'}{T}}$, resulting that a better solution is always accepted, if the difference of the solutions values is positive ($\Delta s, s' > 0$) and the probability of accepting a worse solution is reduced as the distance is higher and particularly as the temperature decreases.

5.4.4 Variable Neighborhood Search

The Variable Neighborhood Search (VNS) metaheuristic (Mladenović & Hansen, 1997) uses cyclically the local search to explore neighbor solutions, starting from a single solution and restarting the search every time a better solution is found from that new solution. In the SearchCol context, two solutions are considered neighbors if their difference occurs in a small number of subproblem solutions. The dimension of the neighborhood is increased by augmenting the number of changes allowed when no more improvements are achieved in the current neighborhood, and the maximum

number of changes is configurable. Details about the VNS implementation in SearchCol are presented in (Alvelos et al., 2013). The steps of a basic VNS are described in Figure 30.

1. Define the neighborhoods used in the search N_k for k=1 ,…, k_{max}
2. Define stopping criterion
3. Set starting solution s as the initial solution s_0
4. **REPEAT** until stopping criterion reached
5. **Set** k=1
6. **REPEAT** until k=k_{max}
7. **Shake**: Select a random solution s' in the neighborhood N_k of s
8. Use local search in solution s' to obtain the local optimum s''
9. **IF** s'' is better than s **THEN**
10. Set s as s'' and k=1
11. **ELSE**
12. k=k+1

Figure 30 – VNS algorithm

5.5 Perturbations

The use of perturbations in SearchCol algorithms was already introduced in Section 4.1. A perturbation is used to fix a subproblem variable to 0 or 1 by adding a new constraint to the RMP, originating that new columns need to be generated to obtain the optimal solution for the new RMP, which is designated as perturbed CG (Alvelos et al., 2013).

In the context of the BDRP, a perturbation can define that a driver does a particular duty, fixing the variable to 1, or that the driver cannot have that duty assigned, fixing the variable to 0. After that, for the case where a duty is fixed to a driver, the RMP can only include in the solution the existing columns generated by the subproblem representing the driver respecting the constraint (including the duty). All the columns from subproblem solutions of the other drivers, that include the duty, cannot have a positive value in the RMP solution.

When a new set of perturbations is included, a new iteration of the SearchCol algorithm starts, including the perturbed CG, to obtain new attractive columns, followed again by the search, which receives a new RMP solution, affecting some of the constructors and that can, if chosen, explore the search space defined by the perturbation or including all the generated subproblem solutions.

As described in (Alvelos et al., 2013), some general perturbation generators are already available, however, it is also possible to define new personalized perturbations in each decomposition model using the knowledge about the problem. We developed a new strategy to define perturbations applied to the BDRP using the knowledge about the structure of the solutions.

5.5.1 Perturbations for the BDRP

The criteria used to select the variables to apply perturbations is very relevant since it defines the number of variables values fixed, which limits the remaining space of solutions and impacts the next CG cycle.

The idea of the developed perturbation generator is to have a greedy behaviour and, in optimal condition, a fixed number of iterations (equals to the number of days in a week) on the SearchCol algorithm.

```
1.  perturbWeekBest(p)
2.      FOR v=1 to number of drivers
3.          FOR week=1 to number of weeks
4.              perturbOnes(v, week)
5.              i = findBest(v, week, p)
6.              IF i >0
7.                  addPerturbation(v, i, 1)
```

Figure 31 – Perturbation generator algorithm

We can define our generator as the function *perturbWeekBest*(*p*) described by the algorithm in Figure 31. The function receives the parameter *p* that defines the lower value of a variable to be considered. It iterates by all the drivers (line 2) and the total number of weeks (line 3) and, for each week and driver, the function *perturbOnes* is initially used to add a perturbation to each variable related to the current driver and week with the value of one. The function *findBest* is used to obtain the index of the variable with the highest fractional value (higher than *p*) in the global solution composed by the subproblem solutions (each subproblem solution as a contribution to the global solution considering the weight defined by the optimal solution of the RMP). If a duty is returned, a new perturbation is created for that duty and the tested driver fixing the variable to 1 (line 7).

In an optimal scenario, the function *findBest* returns a duty of each week of each driver and the corresponding perturbation is added with success. In those conditions, an integer solution is achieved in five iterations (assuming

two obligatory days-off by week). Every time a perturbation is added, one duty is defined, defining the assignment in the corresponding day and driver. Since in each iteration a perturbation is created in each of the weeks, the optimal number of iterations corresponds to the maximum number of days a driver can work in a week.

We propose this perturbations generator to avoid a large number of iterations since our CG cycles consume a large amount of time, however, the optimal scenario is hardly found since, at least two situations avoid the generation of a perturbation:

- At some point, all the duties have very fractional values in the linear global solution, a variable value lower than *p*, and no duty is selected to be fixed;
- A duty is selected but, the function *addPerturbation* fails because there is already a perturbation on the same duty for another driver. E.g.: In the global solution, a duty is assigned to two drivers in the same proportion (50%), *p* is lower than 0,5, and the duty is returned by the function *findBest* for the two drivers. The perturbation is not inserted in the second pair week-driver.

We choose to keep the selection strategy of finding only the best and do not retry and pick the next if a perturbation fails. The result is that a higher number of iterations of the SearchCol cycle is run.

5.6 Algorithms Outline

Now that all the components and configurations were presented we can summarize the building blocks to design the algorithms based in search by column generation we used in this research to address the BDRP.

To describe all the options available to define a new algorithm, we provide an overview (Figure 32) where the alternatives in each of the main components are represented.

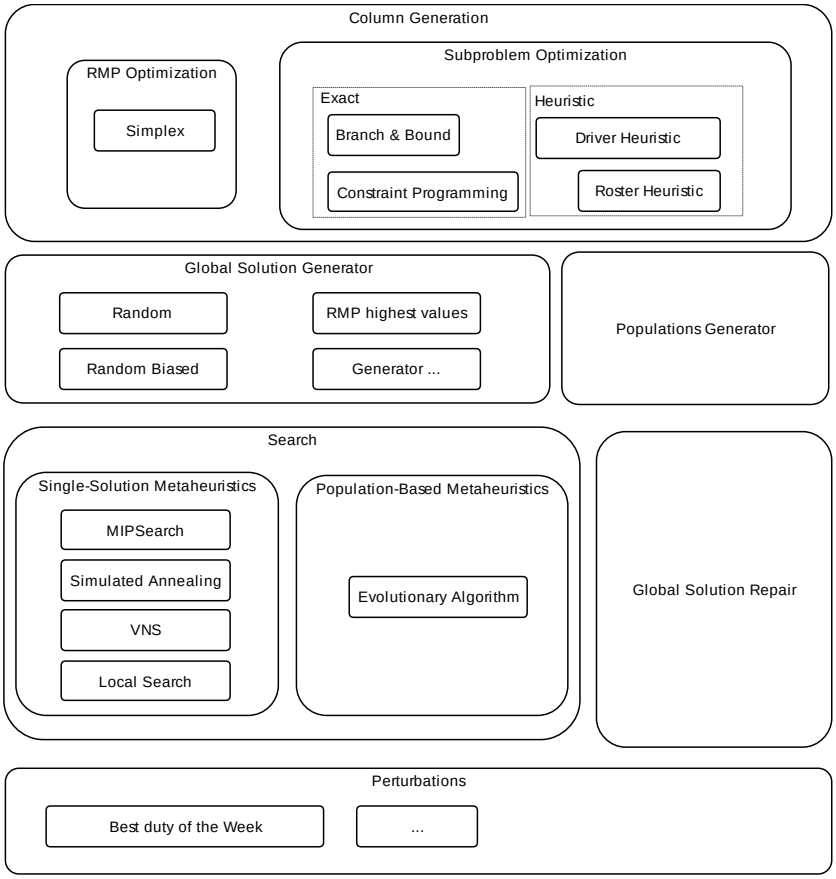

Figure 32 – Components for the algorithms based in search by column generation

The first stage of the algorithm consists in the generation of the search space composed by feasible schedules for all the drivers, as already introduced in Section 5.2.2. The method used to obtain a search space is the use of column generation to solve the decomposition model. The standard column generation cycle can be changed by using the configurations described in Section 5.3.1. Besides that, alternative methods can be selected to solve the subproblems. The branch-and-cut is the exact method to solve the subproblems defined by Model 6 or Model 7. The constraint programming is the method used to obtain solutions for the subproblems when defined by Model 8. If the first two models are used to define the subproblems, one of the two metaheuristics can be selected to obtain feasible solutions for the subproblems in the CG iterations.

Having a search space to explore, and depending on the type of metaheuristic chosen, a single global solution or a population of those

solutions is needed. If a single solution is needed, the generator for that initial solution provided to the metaheuristic must be selected. If the metaheuristic needs an initial population, the single solution generators to be used and the amount of solutions obtained with each one define how the population is created.

If the choice to explore the search space is a single-solution metaheuristic, the alternatives evaluated in our research are presented in Section 5.4 and are also included in the Figure 32. The group of population-based metaheuristics only includes our evolutionary algorithm.

The repair operator we developed is represented in Figure 32 as an independent component as it can be used be any of the metaheuristics and its behaviour goes beyond the search, which was already explained.

If the results obtained by a single search on the initial search space are not satisfactory, the use of perturbations allows to run additional cycles of column generation plus a new search by adding new constraints in the RMP to force the generation of new subproblem solutions, as described in the previous section. Besides the new perturbations generator, developed for the BDRP, other generators exist (Alvelos et al., 2013) (e.g. based on column generation or based in incumbent).

With the presented alternatives, we provide a framework allowing the definition of various algorithms to address the BDRP. The next chapter presents tests with some of the algorithms that can be defined within this framework.

5.7 Summary

In this chapter we described the integration of a new decomposition model (for the BDRP) in the framework which implements the concept of search by column generation, extended with our contributions presented in the previous chapter. We explained how a roster is evaluated within our framework as well as the behaviour of the repair operator developed to repair infeasible rosters.

A section was devoted to detailing the available alternatives in the generation of the search space, and presenting how the column generation can be configured.

One of the configurations available is the use heuristics to obtain subproblem solutions. Two heuristics were presented, which allow to obtain a solution for a subproblem representing a driver or an entire roster by solving all the subproblems combined.

Four single-solution metaheuristics which were used during our research were introduced as was the concept of perturbations where we also proposed a generator based on the knowledge about the BDRP. A final section presented the outline of the algorithms that can be designed by combining a selection of all available components in each stage in order to address the BDRP or other problems, with the needed adjustments.

6 Computational Tests

This chapter is dedicated to the presentation of the computational tests performed during this research.

After the presentation of the instances of the problem used in the tests and the test conditions, the second section describes all the computational tests run in the first stage of our approach, which is the column generation. The alternative configurations of the CG tested are described and their results are presented and discussed. The search-space that results from the best configuration is evaluated by using the MIPSearch metaheuristic in the search for integer solutions and the section ends with the discussion of the results.

The third section describes the computational tests with the single-solution metaheuristics (MIPSearch, VNS and SA) and our basic EA (without the repair operator) in the search for integer solutions in the search space from the best configuration of the CG. The configurations used in each metaheuristic are described and the results from all the tests are presented and discussed.

In Section 6.4, the computational tests with two configurations of our EA (with the repair operator) are described, the results are shown and the section ends with the discussion about the differences between the results of the two configurations tested.

In Section 6.5, additional computational tests are presented to evaluate the solutions obtained in the previous section. Results from solving the problem instances represented by a compact model with a commercial solver are presented and compared with the solutions from our algorithm.

The last section presents the computational tests consisting in the run of multiple cycles of the column generation and the insertion of perturbations obtained by the generator we proposed in Section 5.5. The results from the approach are revealed and discussed.

6.1 Test Instances

The benchmark instances used in the computational tests are divided in three groups. Two result from the solution of Integrated Multi-Depot Vehicle and Crew Scheduling problems, with 80 and 100 trips (p80 and p100) and the instances in the last group (c) were obtained from a real bus company (c), as explained by (Respício et al., 2013). The major distinction between the groups of instances is the size, concretely the total number of duties to assign in the roster. The average number of duties for each group is: p80=467; p100=617; c=856; The exact number of total duties and the details on the number of duties in the week days and weekend days, separated by type (Early/Late), are presented in Table 5. In the instances p80, the number of available drivers is 36 and in the p100 the number is 45. In the instances of group c, the number of drivers available is distinct for each sub-group: c122=25; c224=60; c226=47; c238=60;

Table 5 – Test instances data

	Week		Weekend		
Name	**Early**	**Late**	**Early**	**Late**	**Total**
p80_1	10	8	4	8	456
p80_2	7	7	2	7	352
p80_3	11	7	7	7	472
p80_4	11	7	5	7	456
p80_5	9	8	5	8	444
p80_6	10	6	6	6	416
p80_7	11	8	4	8	476
p80_8	13	9	7	9	568
p80_9	10	8	2	8	440
p80_10	13	11	3	11	592
p100_1	16	12	10	10	720
p100_2	14	11	6	10	628
p100_3	14	12	8	8	648
p100_4	12	11	5	9	572
p100_5	15	11	7	9	648
p100_6	10	11	3	9	516
p100_7	17	9	7	6	624
p100_8	13	13	7	10	656
p100_9	17	11	7	11	704
p100_10	12	7	2	7	452
c122-1	11	6	5	6	428
c122-2	11	6	5	6	428
c122-3	11	6	5	6	428
c224-1	27	14	6	12	964
c224-2	27	14	8	12	980
c224-3	26	15	6	13	972
c226-1	22	12	6	12	824
c226-2	23	12	7	12	852
c226-3	22	14	5	14	872
c238-1	37	12	18	8	1188
c238-2	37	12	15	10	1180
c238-3	38	10	17	8	1160

In each of the instances the duties to assign are the same in each week day (Monday to Friday) and also in the weekend days. The number of duties to assign in each of the instances is presented in Table 5. All the instances consider a four-weeks rostering period, the total number of duties is obtained considering 20 weekdays and 8 weekend days. Each unit of time corresponds to 15min.

In all instances, the parameters used to create the models presented in Section 3.4 are presented in Table 6.

Table 6 – Test instances parameters

Parameter	Value
Rostering period (days)	28
Maximum number of consecutive days without a day-off (g)	6
Minimum days-off by week (d_w)	2
Minimum days-off in a Sunday (in the rostering period) (d_s)	1
Maximum duration of the total of duties assigned to a driver in a week (b_1)	192
Maximum duration of the total of duties assigned to a driver in the complete schedule (b_2)	704
Expected number of days with duties assigned in the driver' schedule (q)	20
Normal duration of a work-day. (before overtime) ($\bar{t}$)	32
Fixed cost of using a driver (C)	100000

The cost of each unit of worktime in the normal work-day (c) is unitary and the additional cost on each overtime unit (ρ) raise from 1 to 8. The set of drivers is divided in four groups of similar sizes and from one group to the next, the cost doubles (1,2,4,8).

All the results presented in this section were made in the same computer. The computer is equipped with an Intel Pentium CPU G640, 2,80GHz, 8 Gb of RAM, Windows 7 Professional 64 bits operating system and IBM ILOG 12.5.1 64 bits.

6.2 Column Generation

As described in Chapter 5, the first stage of our algorithm, and of any other SearchCol algorithm, is the use of the column generation method to solve

the decomposition model and generate the search space to be explored by the metaheuristics.

This section presents the column generation configurations tested and shows the more relevant results from those configurations in solving the test instances.

6.2.1 Alternatives

The standard column generation algorithm iterates between the RMP optimization and the optimization of all the subproblems with the updated objective function.

Considering the three subproblem models presented and the exact solvers available to obtain solutions for the subproblems, the base configurations are presented in Table 7. In addition to these configurations, we can also decide to use one of the proposed heuristics to obtain solutions for the subproblems (*driver* or *roster heuristic*) and, when using heuristics, we can define if the heuristic is used until the end of the CG or if it only used until it is unable to obtain new columns and the following iterations only use the exact solver.

When a heuristic is used with a base configuration, the identification of the heuristic is appended to the name of the configuration as: D, if the driver schedule builder heuristic (Figure 26) is used; RS, if the roster builder heuristic (Figure 29) is used with the sequential rotation of drivers; RR, if the roster builder heuristic is used with the random rotation of drivers. By default, when an heuristic does not obtain a solution to generate a new column, the exact solver is used to obtain the optimal solutions of the subproblems and in the next CG iteration the heuristic is used again. If the CG is configured to, once used the exact solver, do not retry the heuristic, a "+" is appended after the identification of the heuristic. As an example, the name "SP.A.RR+" corresponds to the base configuration SP.A using the roster heuristic with random rotation of drivers and the heuristic is not used again when the exact solver is needed to obtain attractive columns.

Table 7 – Column generation base configurations

Name	Subproblem	Subproblem solver
SP.A	Assignment MIP (Model 6)	Branch-and-cut (CPLEX (IBM, 2016b))
SP.N	Network MIP (Model 7)	Branch-and-cut (CPLEX (IBM, 2016b))
SP.CP	Constraint Programming (Model 8)	Constraint Programming (CP Optimizer (IBM, 2016a))

To avoid longer computational times in the CG, considering the need of a high number of iterations because of the RMP dual solution fluctuation, a time limit of one second was defined to solve each subproblem with the exact solvers. We set the time limit because the subproblem can be degenerated, with multiple equivalent solutions, resulting in difficulties for the solver to find the optimal solution.

6.2.2 Results

Considering the proposed decomposition model for the BDRP and the column generation configurations, distinct search spaces can be obtained from the CG execution. Table 8 shows results from the CG performance in some of the configurations. The CG is tested with the three subproblems structures, starting from the standard CG, which means the subproblems are all solved using an exact solver (columns SP.A, SP.N and SP.CP) and then the standard CG using the *driver heuristic* as an additional subproblem solver (the next three columns ending with D).

After analysing the results of the standard CG with the exact solver and the usage of the *driver heuristic* with the three subproblem models, the *roster heuristic* was initially tested as subproblem solver only in the subproblem assignment model (SP.A), the one with best results in the previous tests.

Two versions of the *roster heuristic* were used, the version with random rotation of the drivers (RR) and the version with sequential rotation of the drivers (RS), in the selection strategy of the next driver to assign a duty. When testing the usage of the *roster heuristic*, independently of the version used, the CG is tested in two configurations: the first, where the heuristic is continuously used until the end of the CG, and the second, where the heuristic is only used until it fails to obtain attractive columns (identified by the "+" in the name of the configurations presented in Table 8).

After the analysis of the results of the usage of the *roster heuristic* in the assignment model, the version with the best behaviour (RR+, random rotation used only until it fails to obtain new columns) was tested in the two remaining models (two last columns: SP.N.RR+ and SP.CP.RR+).

The results presented in Table 8 include the following counts: CG number of iterations, number of SP heuristic runs, number of exact solver runs, number of solutions stored/generated columns and number of instances where the CG time limit was reached; The table also includes the measures of computational time used: total time, the time used by RMP optimization, time used by the heuristic to solve the subproblems and time used by the exact solver to solve the subproblems. The values in the table are average results for each group of instances (*p80*, *p100* and *c*) and for all the instances together.

Comparing the standard CG with the three models (A, N, CP), we observe that the decomposition with the assignment model (A) in the subproblem is the one with best results since the CG could solve all *p80* and *p100* instances in the time limit and the total computational time is clearly better.

Looking at the time used by the SP exact solver, we found the reason for the performance difference between the three models. The integer programming assignment model is solved much faster than the network and constraint programming models.

The increase of the computational time to solve each SP results in a lower number of iterations of the CG in the total time limit, which avoids the achievement of the optimal solution by the CG. In the network model, we observe that, as the instances get bigger (*p80*, *p100* and *c*), the difference on the number of iterations between SP.A and SP.N increases, as well as the number of instances which were not solved in the time limit.

The use of the *driver heuristic* has a significant impact on the number of iterations of CG, increasing the number of subproblems solved and, by consequence, the number of generated columns. The major difference from the standard CG is that most of the time is here spent in the RMP optimization. This can be justified by the increase on the number of iterations (more RMP optimizations), a larger number of columns in the RMP (more time for each RMP optimization) and the lower number of calls

to the exact solver as it is only called when the heuristic is unable to obtain attractive columns.

The decomposition with the network model formulation of the subproblem (SP.N) is the one with the higher improvement from using the *driver heuristic* (SP.N.D) since it could solve more instances and reduce the total computational time in all the instances.

When using the *roster heuristic*, we know that in each iteration of the CG all the subproblem solutions are distinct. The results show that the use of the heuristic until the end of the CG (SP.A.RR and SP.A.RS) decreases the total number of iterations, when comparing with the use of the *driver heuristic* (SP.A.D), but is higher than the number of iterations of the standard CG (SP.A). The same is verified in the computational time, only in the instances of type *c* an improvement is observed from the previous configurations, besides more instances are solved. No significant difference exists between the random (RR) and the sequential rotation (RS) of drivers in the heuristic.

The results of using the *roster heuristic* only in the beginning (SP.A.RR+ or SP.A.RS+), until the first call of the exact solver when the heuristic does not generate attractive columns, are the best for all types of instances. The number of iterations of the CG is reduced, as is the total computational time, and the number of instances not solved in the time limit decreases to 5. Again, the random (RR) and the sequential rotation (RS) of drivers in the heuristic have similar results.

The configuration with the *roster heuristic* only in the beginning of the CG with the assignment model (SP.A.RR+) is the one with better performance and which generates a smaller search space. All the configurations of the roster heuristic were tested in the assignment model. For the two other subproblem models, only the best configuration (RR+) was tested but the results did not improve as in the first model, fault of the time used by the exact solver to obtain the solutions.

Table 8 – Column generation configurations assessment

Config	*SP.A*	*SP.N*	*SP.CP*	*SP.A.D*	*SP.N.D*	*SP.CP.D*	*SP.A.RR*	*SP.A.RS*	*SP.A.RR+*	*SP.A.RS+*	*SP.N.RR*	*SP.CP.RR+*
CG number of iterations												
p80	831.9	806.7	281.8	3479.4	1954.7	3298.1	1080.2	1033.9	542.8	609.2	617.2	393.8
p100	988.4	640.3	215.4	1711.2	1714.1	1740.2	1313.9	1477.6	706.5	707.3	778.9	554.6
c	962.8	287.9	177.3	3000.9	1887.8	2752.1	1148.2	1077.4	715.6	704.7	396.9	385.2
All	929.9	560.2	221.8	2747.4	1854.4	2606.5	1178.7	1188.9	658.8	675.7	585.1	440.8
Number of SP Heuristic runs												
p80	0.0	0.0	0.0	86936.2	45100.2	83071.1	1020.4	952.0	295.5	288.6	288.0	255.1
p100	0.0	0.0	0.0	59673.5	57210.0	59955.9	1306.1	1468.5	473.3	484.8	520.1	431.7
c	0.0	0.0	0.0	108260.	81024.8	101407.5	962.0	897.4	236.0	219.8	239.9	233.8
All	0.0	0.0	0.0	86413.2	62356.2	82723.8	1087.8	1092.9	328.8	324.1	342.5	302.3
Number of SP Exact Solver runs												
p80	22040.2	21049.9	7210.3	2362.9	6905.9	1901.7	1541.7	2123.8	6497.4	8419.1	8675.4	3713.2
p100	34640.1	21793.6	7292.0	771.3	2571.4	1255.1	277.6	311.1	8504.9	8053.0	9320.9	4367.1
c	44751.0	9992.8	7111.9	1554.7	3039.8	1062.5	7894.5	7620.5	22561.8	22894.6	5403.2	5821.9
All	34494.2	17135.9	7198.9	1562.4	4101.6	1384.9	3529.0	3618.6	13148.9	13733.0	7650.0	4708.3
Total time												
p80	710.8	4469.2	7264.3	2668.1	2120.2	4509.1	1237.7	1183.2	**474.2**	525.8	1962.8	3994.1
p100	1721.8	6480.2	7258.0	3032.0	3057.9	4334.1	2928.7	3392.8	1282.1	**1279.0**	4519.7	5240.2
c	5578.3	6841.2	7246.9	6346.1	6282.6	7220.1	5056.4	4673.3	**4797.3**	4822.0	6142.6	7228.3
All	2852.0	5987.1	7255.8	4161.1	3974.1	5471.1	3198.1	3182.5	**2347.9**	2372.2	4329.3	5596.3
RMP optimization time												
p80	471.6	610.1	79.4	2601.3	1062.7	2556.3	1212.9	1152.8	401.3	433.4	470.1	257.6
p100	1291.9	697.6	50.0	2971.9	2188.6	3021.9	2908.8	3373.2	1172.5	1175.9	1459.2	840.8
c	3834.4	139.5	96.9	6193.4	5060.7	6020.4	4819.7	4453.5	4141.1	4137.9	1222.5	1361.6
All	1989.0	461.0	76.8	4064.2	2913.8	4000.8	3095.4	3084.5	2044.7	2054.6	1061.3	853.8
Time used by SP Heuristic												
p80	0.0	0.0	0.0	30.4	18.7	29.4	1.6	1.4	0.4	0.4	0.5	0.3
p100	0.0	0.0	0.0	37.7	38.5	38.2	2.7	3.0	1.0	0.9	1.5	0.8
c	0.0	0.0	0.0	116.4	116.5	115.0	4.2	3.8	1.0	0.9	1.6	0.9
All	0.0	0.0	0.0	64.9	61.6	64.2	2.9	2.8	0.8	0.7	1.3	0.7
Time used by SP Exact Solver												
p80	234.8	3834.2	7137.2	26.8	998.7	1911.6	18.0	23.8	69.5	88.8	1436.3	3734.4
p100	420.0	5758.0	7205.7	9.5	763.5	1262.0	3.2	3.6	102.1	95.6	2925.7	4395.5
c	1738.1	6682.6	7124.8	24.4	1003.7	1069.3	221.1	204.2	646.8	675.7	4896.4	5863.9
All	856.4	5503.5	7153.9	20.5	927.1	1392.7	89.5	85.1	296.2	311.0	3199.3	4739.6
Number of Subproblem Solutions (Generated Columns)												
p80	21480.0	20303.6	7288.9	61038.5	30505.6	58710.5	28841.5	27420.4	13273.4	14378.4	14804.5	9748.5
p100	34256.4	21678.0	7395.5	56239.8	44956.3	55401.6	43968.4	49433.5	22971.5	23147.0	25634.0	17833.7
c	44660.1	9986.3	7249.7	87782.9	68468.3	83893.6	52663.9	49139.3	32429.5	32164.4	16210.9	16397.5
All	34165.2	16864.1	7307.5	69568.1	49257.5	67120.1	42502.1	42444.1	23487.6	23788.3	18716.1	14768.5
Number of Instances where CG stop by time limit												
p80	0	2	10	1	0	1	1	1	**0**	**0**	0	0
p100	0	8	10	0	0	2	0	2	**0**	**0**	1	3
c	9	10	12	9	9	12	6	5	**5**	**5**	9	11
All	9	20	32	10	9	15	7	8	**5**	**5**	10	14

Besides the aggregated results presented in Table 8, detailed results about the individual tests of each instance are presented in Appendix A.

Appendix A also includes charts presenting the solution value evolution (y axis) through optimization time (x axis), in seconds, for all the instances in the best configuration (SP.A.RR+).

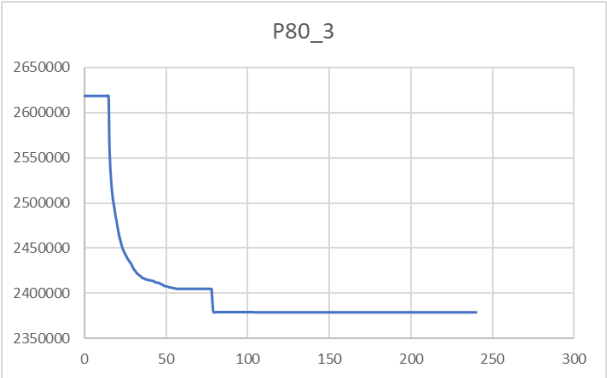

Figure 33 – Example of CG solution evolution

Figure 33 displays an example of the evolution of the solution value in the CG. The tail-off effect, usual in CG, is clearly visible. In most of the instances we observe a descending curve on the solution value and then a step to a value closer to the final value, from where no significant decrease on the solution value is observed. The initial descending curve corresponds to the use of the roster heuristic and the larger step to the first execution of the exact solver when the heuristic is unable to obtain more attractive columns. After the first improvement, the solutions obtained by the exact solver allow very small improvements until the end of the CG.

6.2.3 Search Space

To evaluate the search space resulting from solving the decomposition model with column generation, a searcher (MIPSearch) from the SearchCol framework was used.

To test the search space obtained by the column generation, the MIPSearch sets all the variables associated to the generated columns as binary, to assure that each driver will be assigned to only a schedule (subproblem solution used to add the column in the RMP) and not to a convex combination of schedules as in the optimal linear solution of the RMP. To obtain the optimal solution available in the search space, which is the set of all

schedules created during the CG, the RMP is optimized as a MIP by the solver using the branch-and-cut method.

Table 9 presents the results obtained after 1800s of execution of the MIPSearch. The table show only the infeasibility value achieved in each instance. The infeasibility value if obtained considering the formula presented in Section 5.2.1, considering a cost of 1000 for each unassigned duty (parameter inf_{c1}) and a cost of 10 for each over-assigned duty (duty assigned to more than one driver) or extra days-off in the work-schedules (parameters inf_{c2} and inf_{c3}). A global solution is feasible (with all duties assigned) if the infeasibility value is lower than 1000 (the cost of an unassigned duty). The MIPSearch reached a feasible solution only with two instances (p80_9 and p100_1).

In the infeasibility value obtained in the feasible solution for the two instances, each amount of 200 in the value corresponds to an additional driver beyond the minimum number of drivers possible defined by the equation (74), as each driver can have a maximum of 20 duties assigned, or extra days-off and each additional day-off or duty over-assigned is penalized by the cost of 10.

Table 9 – MipSearch infeasibility value results.

Instance	Infeasibility Value	Instance	Infeasibility Value	Instance	Infeasibility Value
p80_1	100360	**p100_1**	400	**c122_1**	35280
p80_2	55480	**p100_2**	110560	**c122_2**	67520
p80_3	83020	**p100_3**	134220	**c122_3**	63640
p80_4	87300	**p100_4**	94440	**c224_1**	281500
p80_5	72000	**p100_5**	146460	**c224_2**	281700
p80_6	67720	**p100_6**	77100	**c224_3**	288440
p80_7	86080	**p100_7**	179100	**c226_1**	248260
p80_8	111780	**p100_8**	159920	**c226_2**	243560
p80_9	600	**p100_9**	124220	**c226_3**	261720
p80_10	121980	**p100_10**	50980	**c238_1**	355980
				c238_2	355160
				c238_3	359440

A first conclusion from the results presented in Table 9 is that the solver spent all available time (1800s) applying branch-and-cut to search for the optimal solution, indicating that it is time consuming to completely explore

the search space finding the best available solution in the set of all available schedules.

The infeasibility values also show the difficulty of finding a set of subproblem solutions covering all the duties in the original search space. This may occur because the optimal linear solution of the CG (final RMP) is very fractional, indicating that each duty is present in many schedules, resulting that, as more schedules are fixed in the branches of the branch-and-cut, harder it becomes to find complementary schedules assigning the remaining duties, without including repeated assignments (which are occupying space in the drivers' schedules).

The results also show that a more efficient search method is needed to explore the available search space, since this exact method consumes too much time.

6.2.4 Discussion

The results of the column generation configurations tested in this section show that the choice of the pricing problem and the combination of subproblem solvers has a significant impact in the performance of the method. Even with the best identified configuration, the "tail-off" effect makes that the method takes a significant time to reach the optimum linear solution.

The results of the MIPSearch heuristic in the search-space resulting from the more efficient CG configuration show the difficulty of finding (even feasible) solutions by solving directly the RMP with a MIP commercial solver. One of the reasons for the bad solutions provided by MIPSearch may be the size of the search-space, since when the CG runs more time a larger number of columns are generated, making the complete search harder. Another reason is the fact that the optimal linear solution of the RMP is very fractional, thus avoiding the identification of good components for the global integer solution. When using the *roster heuristic*, some of the solutions of the heuristic are feasible rosters, and since the MIPSearch is unable to achieve feasible solutions for most of the instances, this demonstrates that the linear solutions are far from an integer solution and they do not provide a clear direction to feasible integer solutions.

In the current scenario, the normal path, already available in the SearchCol framework, is to run additional cycles of column generation to obtain additional schedules by adding perturbations fixing partial solutions. The use of perturbations has an obstacle in the BDRP which is the amount of time consumed by the CG cycles. Since, the CG is currently the part of the algorithm where most of the time is used, we search for alternative paths avoiding the need of additional CG cycles which were introduced in Section 4.1.

6.3 Metaheuristic Searcher

Besides the results of the MIPSearch on the original search-space, indicating the difficulty of reaching good quality solution with a single search, in this section we present the tests with two single solution metaheuristics, simulated annealing (SA) and variable neighbourhood search (VNS), previously described in Section 5.4, and with our evolutionary algorithm (EA), in the version without the repair operator.

The main objective of these tests is to evaluate the EA, the first population-based metaheuristic integrating the SearchCol framework is a competitive alternative as a metaheuristic for search in comparison with pre-existing single-solution metaheuristics.

6.3.1 Configurations

In the VNS configuration:

- The neighbourhood size can grow from 1 to 5, with steps of 1;
- The local search was configured to stop at the first improvement found;
- The maximum number of searches without improvement is 100, which is a stopping criterion.

The SA was configured with:

- A probability of a worse infeasible (greater infeasibility value) solution to be accepted initialized to 20%;

- A probability of a worse feasible (solutions with identical or null infeasibility value) solution to be accepted initialized to 90% of the occurrences;
- The initial temperature is defined by the average variation on the evaluation values between the initial solution and each of its neighbours divided by the logarithm of the probability of acceptance (which changes if the initial solution is feasible or not);
- The temperature decrement (alpha) was set to 0.95;
- The initial number of iterations in each temperature is 100 and is increased by 5% in each temperature update;
- The minimum temperature was set to 10, which is the stopping criterion.

For the VNS and the SA, the initial solution is built by selecting, for each subproblem, the solution with the higher value from the optimal linear solution values in the last CG optimization.

The EA needs an initial population, which is built using the existing generators for global solutions, as explained before in Section 4.2.1. The initial population generated for the EA contains a global solution obtained from the generator that selects the solution from each subproblem with the higher value in the optimal solution of the RMP (G1) and another composed by the last optimal solutions from each subproblem (G2). The remaining individuals included in the population are created by generators with stochastic behaviour. The first one (SG1) selects randomly each subproblem solution. The second (SG2) selects randomly the solution but the probability of each solution is biased by the value of the solution in the solution of the RMP, which results in considering only the solutions that are included in the linear solution of the CG. The number of individuals produced by these generators depends on the number of subproblems. The first generates the number of subproblems multiplied by 6 and the second the number of subproblems multiplied by 10. Depending on the instances, the estimated population size is between 300 and 1000 individuals.

Our EA is configured to use the binary tournament as selection operator, the one-point crossover is use in 80% of the individuals included in the mating pool and the remaining go to the next generation without change. The

probability of using the mutation operator in each individual included in the next generation is set to 20%, allowing to explore the search space which is not part of the initial population. An elite pool with the 30 best individuals is kept along iterations from where 10% of the mating pool individuals are selected. In each interval of 10 generations of the EA, a random individual is selected and the local search heuristic is used to try to improve it. If the number of generations without improvement reaches 99% of the limit value, the local search is used in all generations on the best solution found. In the initial test, the repair operator is not used.

The stopping criterion for the EA was set to 500 generations without improvement on the best solution.

6.3.2 Results

Each metaheuristic (EA, SA and VNS) was tested by running 30 times for each instance. The results also include the results of the MIPSearch which was run only once for each instance, since it is an exact method.

The comparison on the best solution found with each metaheuristic in each instance is presented in Figure 34 for the set of instances P80, in Figure 35 for the set of instances P100 and in Figure 36 for the set of instances C. The charts display the number of unassigned duties. For the EA and the MIPSearch the values are represented by columns and the values from the SA and VNS are represented by markers (a dash for SA and a cross for VNS) allowing an easier comparison in each instance.

The distribution of the best solution found by each metaheuristic for all tested instances is collected in Table 10. The SA is clearly the metaheuristic achieving the best solution more times, with a count of 25, the EA is the second, with a count of 5, highlighting that 4 are from the set of larger instances (all the C224 and one C238), the VNS reached the best solution 4 times and the MIPSearch only 2 times, corresponding to instances where it found feasible solutions, without duties not assigned (P80_9 and P100_1).

The sum of the values in Table 10 is higher than the number of instances because, in some instances, two metaheuristics reached the same solution. The exact same infeasibility value was obtained when a feasible solution

was found (P80_1, reached by EA and SA; P80_2, P80_6 and P80_8, by SA and VNS).

Table 10 – Count of best solution found by each metaheuristic

EA	MIPSearch	SA	VNS
5	2	25	4

In Figure 34, it is visible that for instance P80_1 the EA and the SA have the same value (zero), and also the three occurrences where the SA and VNS were simultaneously able to obtain the same value (P80_2, P80_6 and P80_8).

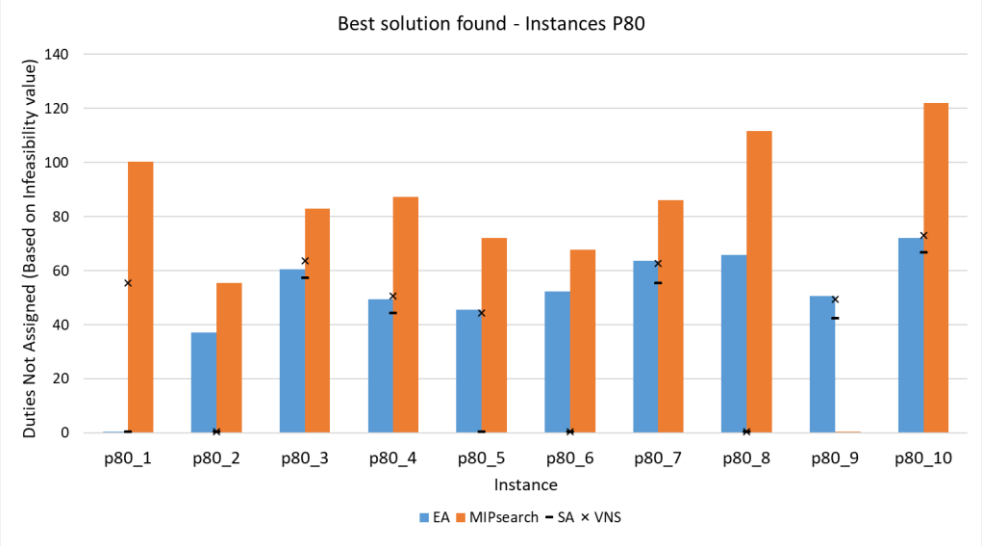

Figure 34 – Best solution found, instances P80

In Figure 35, a feasible solution was achieved only two times (P100_1 by MIPSearch and P100_5 by SA). With exception to the situations where a feasible solution was found, the differences between the EA, SA and VNS are small (an average difference of seven duties). Globally, the three metaheuristics overcome MIPSearch.

The solution values for the larger instances in Figure 36 (subsets C224, C226 and C238) seem even closer between the EA, SA and VNS, and, in fact, the average difference is eight duties. In those instances, the difference for the results obtained by MIPSearch are more evident.

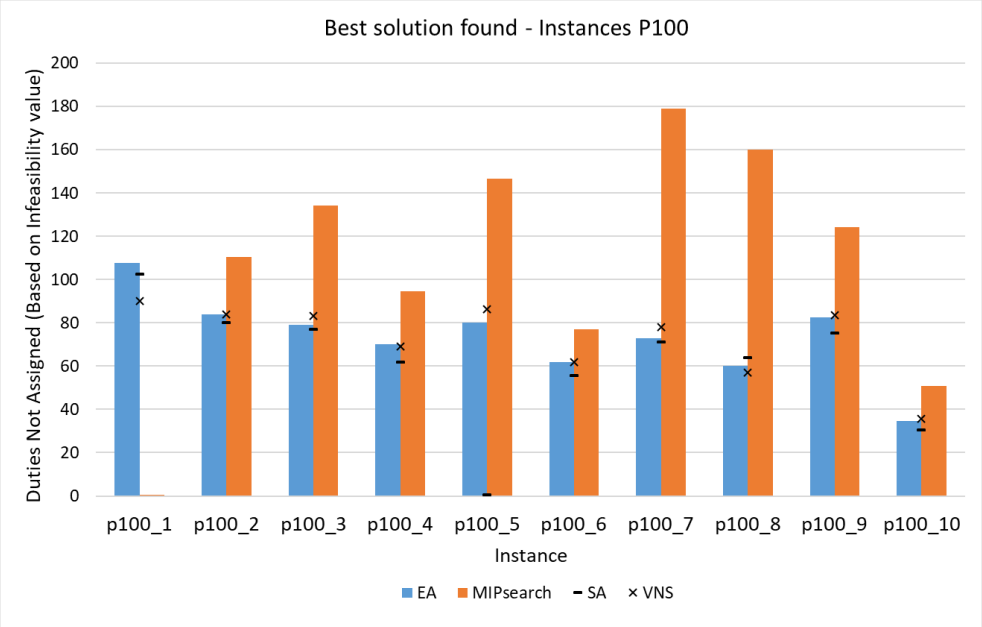

Figure 35 – Best solution found, instances P100

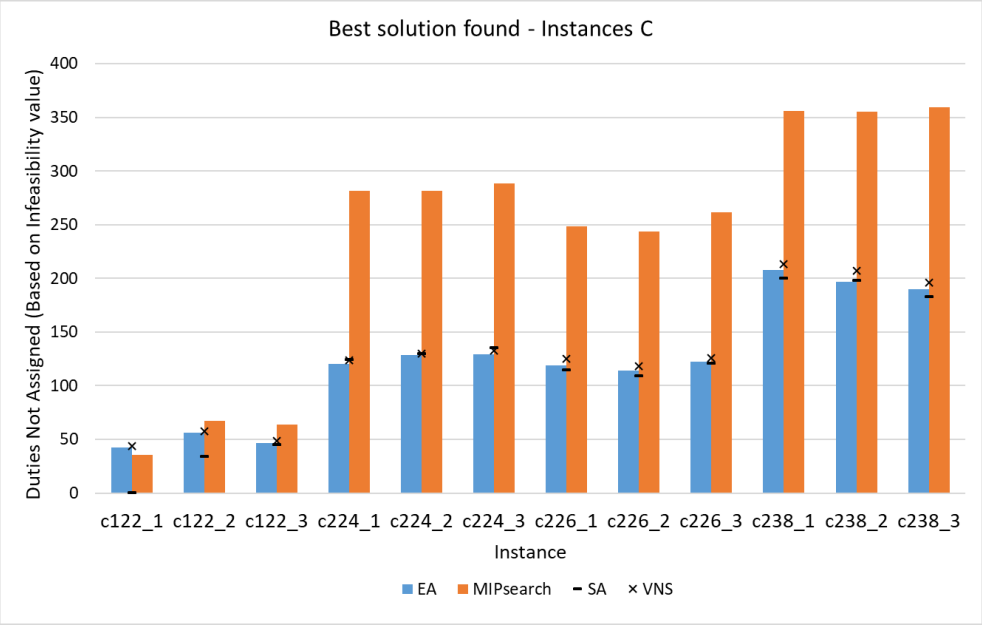

Figure 36 – Best solution found, instances C

The next set of figures (Figure 37, Figure 38 and Figure 39) present the average value of the 30 runs of each metaheuristic (EA, SA and VNS). The results of the MIPSearch are the same of the previous figures because it was run just once.

In Figure 37, the domination of the SA as the metaheuristic reaching the best results is evident, as it has the lower value for all instances. With exception of the instance P80_6, the average difference between the EA, SA and VNS is nine duties.

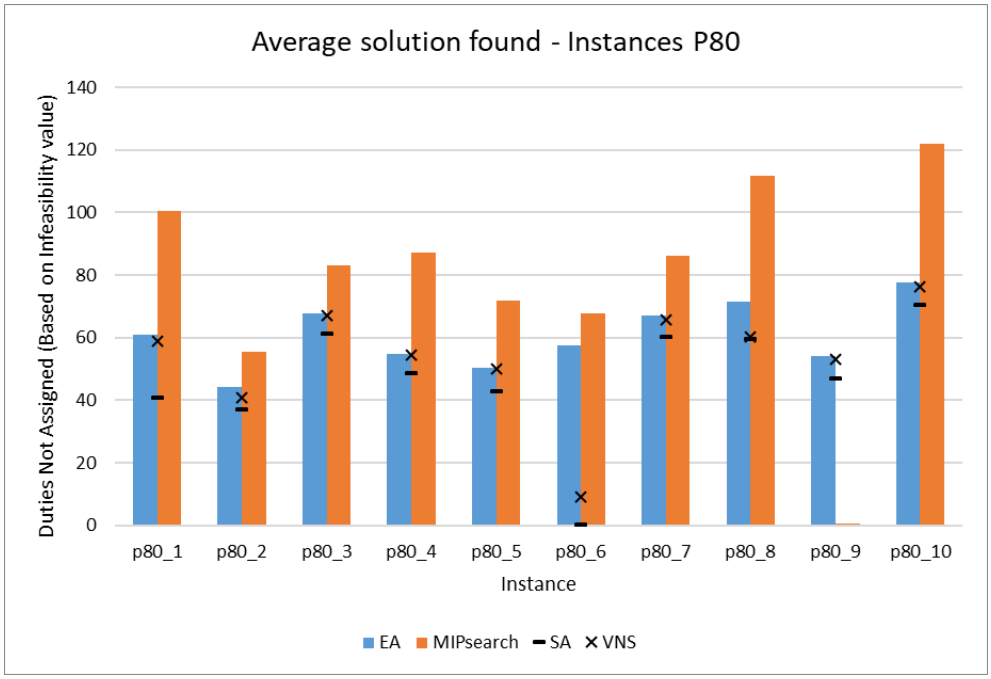

Figure 37 – Average solution found, instances P80

As occurred with the best solution found, as the instances get bigger, the differences on the average values from EA, SA and VNS decrease. Neglecting the worse result of the EA on P100_8, visible in Figure 38, the average difference on the three metaheuristics is seven, and if that result is considered, the difference continues to be nine, as in the set P80.

Again, in Figure 39, the results appear closer, in part due to the larger numbers on the axis, but the average difference is eight duties, which, considering the average number of 129 duties not assigned in this set of instances, corresponds to an average difference of 6%.

We highlight that, for the three larger subsets of the instances C (C224, C226 and C238), the EA has always better results than VNS. The EA continues to be the metaheuristic with best results on the instances C224_1 and C224_3.

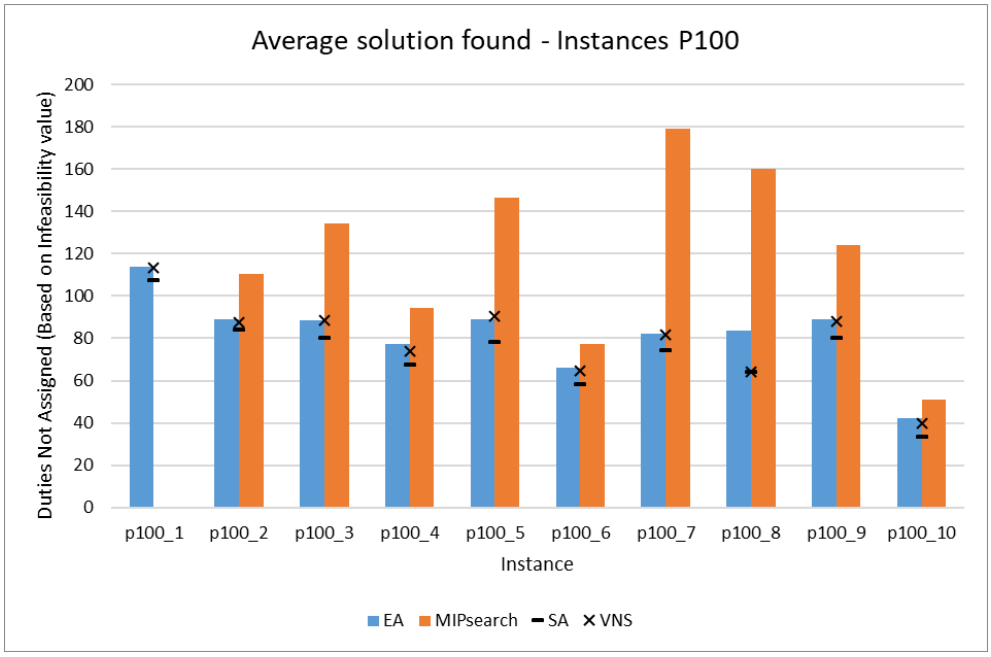

Figure 38 – Average solution found, instances P100

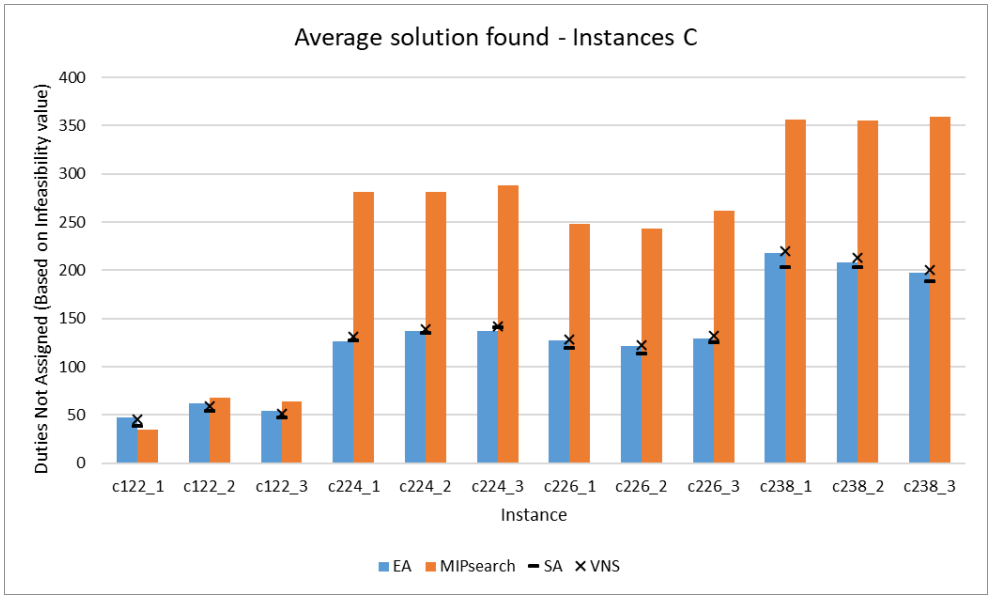

Figure 39 – Average solution found, instances C

The previous figures have presented the best and average solution values achieved by each of the four metaheuristics tested. Figure 40 compares the average time spent by each metaheuristic in the search for the best solution.

With exception to the MIPSearch, with a time limit of 1800s, always used, in the remaining metaheuristic values, we observe that the time used is dependent on the instance size, which is normal, and is particularly coherent is the evolution of the values of the EA and the SA observed in Figure 40. The VNS is the one where the increase of the instance size results in a larger increase in the search time so, for the bigger instances (C224, C226 and C238), a time limit of 600s was set, which is approximately the maximum time used by SA and EA in that subset of instances.

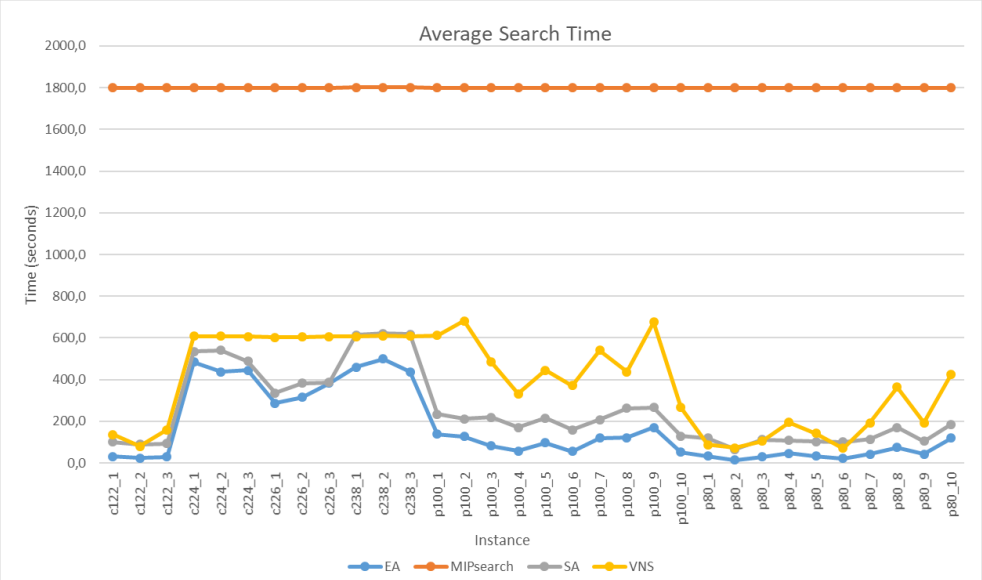

Figure 40 – Metaheuristics average search time

The results presented in Figure 40 show that the EA configuration tested is the fastest metaheuristic reaching the solution, followed by the SA and then the VNS. The average time of the EA search is 166 seconds, the SA search, 252 seconds, and the VNS, 392 seconds.

To describe better the behaviour of our EA as a searcher, Table 11 shows aggregated data from the 30 runs for each instance and also for each set of instances when the values are relevant. The table includes the average infeasibility value of the solutions found and also the value of the best solution found in all runs, it also includes the average improvement on the best solution, considering the best solution included in the initial population and the final one, the average search time and iterations count.

Table 11 – Evolutionary algorithm results

Instance	Average Infeasibility	Minimum Infeasibility	Average New Assignments	Average Time	Average Generations
p80	**60595**		**73**	**46,0**	**1200**
p80_1	60920	400	57	33,0	1211
p80_2	44192	37120	55	15,1	1011
p80_3	67652	60580	72	29,9	1196
p80_4	54830	49560	83	45,5	1218
p80_5	50512	45480	76	33,8	1213
p80_6	57384	52420	51	22,7	1069
p80_7	67176	63640	70	42,0	1185
p80_8	71592	65880	99	74,3	1286
p80_9	54150	50580	64	43,3	1192
p80_10	77542	72000	107	120,5	1421
p100	**82124**		**103**	**102,5**	**1206**
p100_1	113824	107500	108	138,9	1484
p100_2	89072	84040	108	127,7	1050
p100_3	88728	79140	102	82,8	1063
p100_4	77304	69960	94	57,9	945
p100_5	89170	80160	110	97,3	1360
p100_6	65880	61800	100	56,5	1005
p100_7	82234	73020	103	119,2	1393
p100_8	83760	59960	89	121,9	1305
p100_9	88826	82400	124	170,3	1336
p100_10	42446	34660	93	52,8	1124
c122	**54335**		**70**	**27,8**	**1082**
c122_1	47282	42420	72	30,4	1172
c122_2	61740	56300	68	23,4	1038
c122_3	53983	46300	70	29,4	1035
c224	**133747**		**169**	**455,0**	**1374**
c224_1	126800	120340	175	484,3	1337
c224_2	136894	128700	159	436,4	1335
c224_3	137548	129320	171	444,3	1448
c226	**125969**		**141**	**328,5**	**1634**
c226_1	127016	118720	135	287,6	1677
c226_2	121772	114020	141	314,4	1545
c226_3	129120	121980	148	383,5	1681
c238	**208053**		**154**	**465,4**	**1358**
c238_1	218212	208080	146	459,5	1476
c238_2	208280	197060	149	498,8	1353
c238_3	197668	190120	167	438,0	1245
Global	**93547**		**105**	**166,1**	**1263**

The more relevant information in Table 11 is the column with the average new assignments (duties assigned by the EA), stating that the EA could assign 105 new duties on average, even if around 93 duties are unassigned in the average results. This also shows that the EA is able to assign more than 50% of the unassigned duties is the best initial solutions.

Additional data from the results is included in Appendix B.

6.3.3 Discussion

The comparison of the results of the four metaheuristics (MIPSearch, EA, SA and SNV) presented in this section shows the difficulty of reaching good solutions when exploring the search space resulting from the complete column generation execution (or the maximum of 2 hours).

Except for the instance P80_6, where the average value of unassigned duties in the results of the SA is zero, none of the three metaheuristics with 30 runs (EA, SA and VNS) for each instance, could regularly achieve feasible solutions.

The value of the best solution found by the EA considering all the runs are promising, especially the detected improvement of the results as the instances get bigger, reaching the best solution for 4 out of 9 of the larger instances (Instances C).

The EA average results are slightly worse than the other two metaheuristics (SA and VNS), in particular for the smaller instances, but it is justified by the fact that the initial population of the EA is composed in majority by individuals built in a random way, even if some of the probabilities are biased by the RMP optimal linear solution, and the initial solution of the SA and the VNS is built by selecting the subproblem solution with the higher value in the RMP optimal linear solution, resulting that those metaheuristics start always from a good point, according to the column generation solution.

A deeper look on these results and the corresponding solutions allow to detect that the main problem to reach a good solution is the amount of duplicated duties assigned in a global solution (roster) and the consequent difficulty of the metaheuristics in replacing a work-schedule by a new one containing missing duties and the ones in the original work-schedule that are not duplicated.

When removing a work-schedule, the duplicated duties are dropped, but also those that are included in that work-schedule and are unique in the entire roster. The identification of this problem was the trigger to the development of the repair operator.

6.4 Evolutionary Algorithm

Preliminary tests presented in Section 6.3 revealed that the basic EA can improve the global solutions (from best solution in the start of the search to the best in the end) but has difficulties to reach feasible solutions. One of the reasons for the poor results is the fact that the optimal linear solution of the CG is very fractional, avoiding that the generators get good initial solutions (even the one which picks the solution with the highest value in the optimal linear solution). The second reason is that, the algorithm is unable to find the subproblem solutions it needs to assign the remaining duties, without losing those already assigned and replacing the over-assigned ones. That motivated us to develop the repair operator to be integrated in our EA to overcome these difficulties in reaching feasible solutions. The tests of the EA using the repair operator are presented in this section.

To define the search space to be used by the EA (with the repair operator), the CG with configuration SP.A.RR+ was repeated for the set of instances C with the time limit of 1800 seconds, since we intended to obtain results in the time limit of one hour, half time in the CG and other half time in the Search, maximum. When the CG reaches the 1800 seconds the subproblems are already been solved by the exact solver (the last significant descent before the tail-off) and with small improvements (can be observed in the charts in Appendix A).

6.4.1 Configurations

The EA configuration continues using the binary tournament as selection operator, the crossover is used in 80% of the individuals and the mutation operator in 20%. An elite pool with the 10 best individuals is kept along iterations from where 5% of the mating pool individuals are selected. These parameter values were selected from preliminary tests.

The stopping criterion was set to 200 generations without improvement on the best solution or reach the time limit for the search of 1800 seconds. The local search was disabled because it doesn't have any substantial impact when used with the repair operator.

We set the usage of the repair operator every 20 generations in the individuals that have a positive infeasibility value.

The initial EA population continues to include global solutions obtained from the generators G1, G2, SG1 and SG2 as in the previous tests. The number of individuals produced by the stochastic generators (SG1 and SG2) depends on the number of subproblems and the configuration.

Tests of two configurations of the EA were run (C1 and C2). In the first (C1), the initial population is larger and includes the global solutions from G1 and G2, 6 times the number of subproblems generated using SG1 and 10 times the number of subproblems generated using SG2. In the second (C2), we no longer use the generator which selects the subproblem solutions randomly and the number of individuals generated by the other stochastic generator (SG2) was set to 10 times the number of subproblems for the instances *p80*, *p100* and *c122*, as in C1, but was decreased to 4 times the number of subproblems for the remaining *c* instances (with a larger number of subproblems/drivers).

The configuration C2 was tested because the repair operator has a substantial computational burden, and we want to compare the improvement in the search time and in the solutions obtained using a smaller and more constant population size. Besides the smaller size, the initial population used in C2 is obtained considering only the subproblem solutions that belong to the linear solution of the CG.

In C1, the average size of the population is 540, with the maximum size around the 1000 individuals, for the larger instances of group C. In C2, the average size of the populations is 250 individuals, with the size of the population used in the larger instances of group C below the average.

6.4.2 Results

The results of the new tests, with 30 runs for each instance, are presented in Table 12. For both configurations, all the obtained solutions were feasible for all runs. For each instance and for both configurations, the table presents the best solution found (solution value, corresponding to the feasibility value), the average solution in all the runs, the average time in seconds, average number of assignments (difference in the number of duties assigned between the best solution in the initial population and the final solution) and the average number of iterations of the EA.

Table 12 – EA with repair results

	Best Solution Value		Average Solution Value		Average Time		Average # Assignments		Average # Iterations	
Instance	**C1**	**C2**	**C1**	**C2**	**C1**	**C2**	**C1**	**C2**	**C1**	**C2**
p80_1	**2316543**	2316564	2330017	**2316645**	27,8	13,9	112	117	375	361
p80_2	**1812692**	**1812692**	**1812738**	1812739	18,3	11,7	98	98	488	545
p80_3	2419156	**2419149**	**2422591**	2425963	40,4	15,8	140	140	490	398
p80_4	**2316440**	2316454	2403635	**2370114**	42,4	20,2	137	138	411	407
p80_5	2315206	**2315149**	**2315256**	2315260	110,7	68,6	126	127	830	985
p80_6	**2115818**	2115836	2119243	**2115913**	21,3	10,1	111	108	382	361
p80_7	**2417442**	2417450	**2487755**	2501093	45,4	20,5	138	138	443	402
p80_8	**2920965**	2920980	2944538	**2924451**	150,4	64,0	171	171	635	597
p80_9	2216393	**2216368**	2387016	**2323384**	70,6	17,6	94	92	590	539
p80_10	**3019911**	**3019911**	3026759	**3020124**	437,1	159,4	185	186	970	896
p80	2387056,6	2387055,3	2424954,8	2412568,6	96,4	40,2	131,2	131,5	561,4	549,1
p100_1	**3620855**	3620881	**3620953**	3621007	1020,0	364,2	222	222	1084	944
p100_2	**3219732**	3219763	**3219823**	3219839	741,0	409,3	197	197	1200	1373
p100_3	**3319154**	3319177	**3319235**	3319281	1135,1	458,8	190	190	1455	1447
p100_4	2918149	**2918118**	**2918208**	2918232	406,8	164,4	171	171	1025	967
p100_5	**3319564**	3319580	**3319635**	3319666	867,5	328,2	198	199	1274	1183
p100_6	**2615214**	2615225	**2615271**	2615289	375,2	140,3	167	167	1138	1020
p100_7	**3216309**	3216341	**3216361**	3216393	1367,5	462,1	185	185	1666	1482
p100_8	**3322291**	3322355	**3322450**	3322470	705,6	291,0	173	173	983	964
p100_9	**3622050**	3622086	**3622221**	3622252	1132,2	464,2	212	212	1144	1131
p100_10	**2313365**	2313377	**2330110**	2380212	189,4	66,9	137	136	895	765
p100	3148668,3	3148690,3	3150426,7	3155464,1	794,0	314,9	185,2	185,2	1186,4	1127,6
c122_1	**2216755**	2216899	**2217033**	2217073	54,9	26,5	118	119	647	671
c122_2	**2217951**	2217957	2218163	**2218147**	37,4	18,7	130	130	556	572
c122_3	2219435	**2219424**	2229822	**2226434**	42,3	15,3	123	124	533	411
c122	2218047,0	2218093,3	2221672,7	2220551,3	44,9	20,2	123,7	124,3	578,7	551,3
c224_1	**4940156**	4940172	**4940385**	4940477	1231,9	94,3	278	278	814	624
c224_2	**5244932**	5244975	**5245134**	5245137	1266,7	120,5	124	121	698	694
c224_3	**4939380**	4939414	4946252	**4942965**	1009,3	75,8	231	231	786	586
c224	5041489,3	5041520,3	5043923,7	5042859,7	1169,3	96,9	211,0	210,0	766,0	634,7
c226_1	**4239915**	4339894	4357092	**4340291**	262,4	33,2	122	118	437	396
c226_2	**4440727**	4440971	**4468057**	4468176	433,8	39,9	124	123	512	427
c226_3	**4440854**	4442204	**4532422**	4532656	334,9	53,1	141	163	418	463
c226	4373832,0	4407689,7	4452523,7	4447041	343,7	42,1	129,0	134,7	455,7	428,7
c238_1	**6052970**	6053113	**6053308**	6053492	1774,9	206,8	344	344	780	808
c238_2	**5952222**	6051728	**6041940**	6051908	1333,8	123,2	176	143	553	471
c238_3	**5951162**	5951287	**5951400**	5951489	1289,3	99,2	149	148	592	452
c238	5985451,3	6018709,3	6015549,3	6018963	1466,0	143,1	223,0	211,7	641,7	577,0
all	3381678,4	3387984,2	3404838,2	3402142,9	561,8	139,3	163,3	162,8	775,1	729,4

In the columns with solutions values, the best between each configuration is highlighted with bold. The number of highlighted solutions is higher for C1 but, for the best solution, neglecting the instances c226_1 and c238_2 where C1 found a solution with one driver less, the average difference is only 77.

Considering the average values, C1 has again the best value in a higher number of instances, however, the average for all instances for C2 is lower by 2610 in comparison to C1.

The biggest difference observed in Table 12 between C1 and C2 is in the average search time, where C2, with 139 seconds, uses only 25% of the time consumed by C1, with an average time of 562 seconds for the EA execution. Part of the search time reduction results from the decrease of the population size from C1 to C2, however, the reductions are not proportional since, for the instances p80, p100 and c122 the population size in C2 was decreased to 63% of the population size in C1 but the average time spent by C2 corresponds to only 40% of the time used by C1 in this group of instances. In the remaining instances of the group c, the population size in C2 was decreased to 25% of the population size in C1, but the average time spent by C2 corresponds to less than 10% of the time used by C1.

The average number of iterations run has a difference of 5%, with C1 running 775 iterations on average on each execution and C2 729 iterations. The average time by iteration is 0,68 seconds for C1 and 0,15 seconds for C2.

The observed differences in the search time of C1 and C2 are the result of decrease on the population size, possibly helped by having reduced the domain of subproblem solutions to the ones present in the linear solution of the CG by using only the generators G1, G2 and SG2, however, the results also reveal that some instances have unknown characteristics which impact the search time.

The chart displayed in Figure 41 represents the increase of the population size (which is dependent on the number of drivers) in the instances tested and the corresponding search times for both configurations. It is evident that in some instances (e.g. p100_7 or c238_1) the search time has a substantial increase which is not justified by the size of the population and/or instance and is observed in both configurations, although softened in configuration C2. In the chart we can also observe that in the group of instances c226, which has the third larger population size, the search ends in less time than most of the instances from the group p100, which population size is smaller. The difference is more evident in the configuration C1.

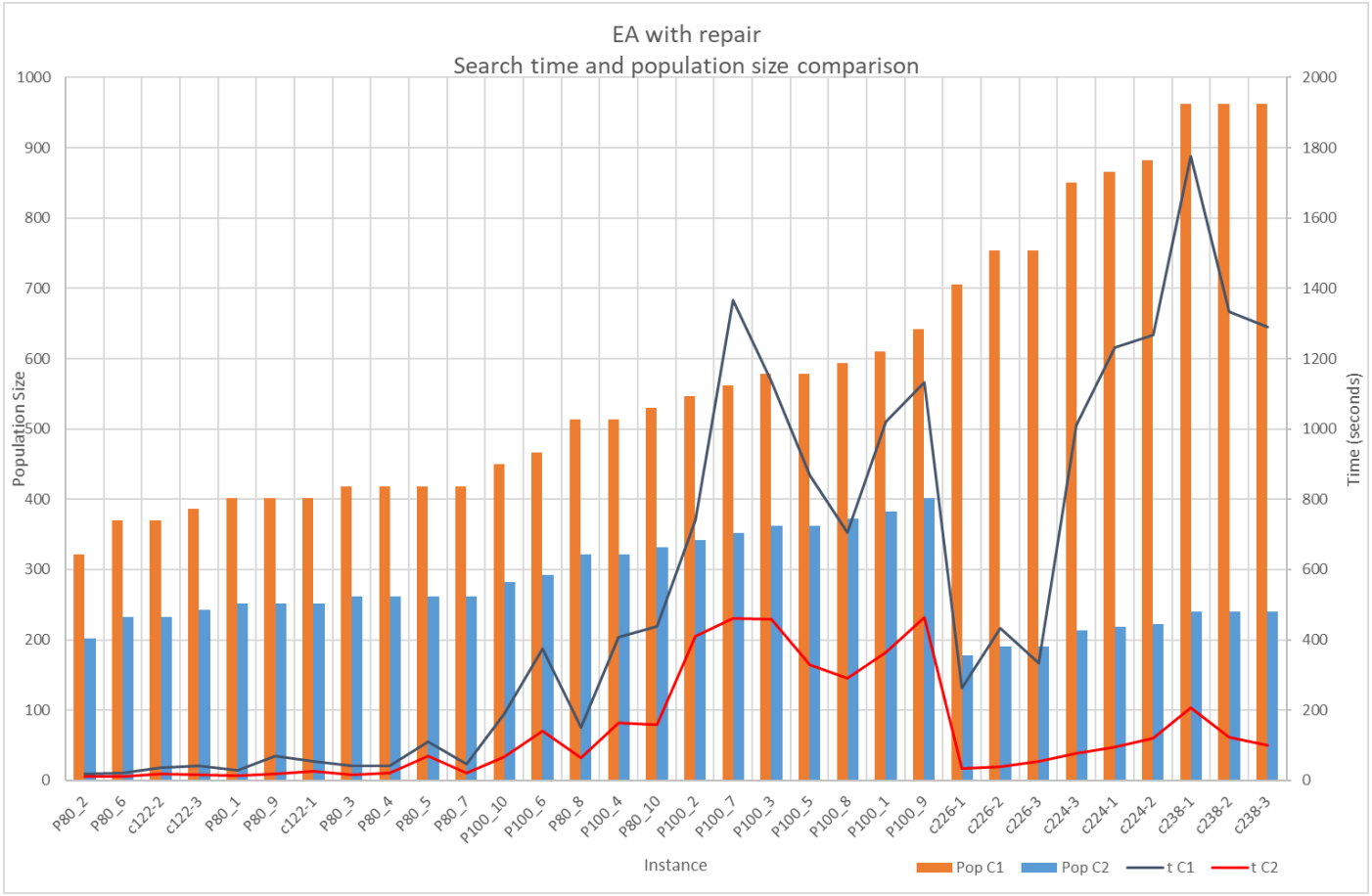

Figure 41 – Search time and population size comparison

For each instance, the repair operator only uses the number of drivers corresponding to the number obtained by the equation (74), however, in some instances, additional drivers were allowed to obtain feasible solutions in all runs. For the instances c226_1, c226_2, c226_3, c238_2 and c238_3 the repair operator could use the value define by the equation (74) plus one, but for the instance c224_2, three additional drivers were needed.

Additional data from the results is presented in Appendix C.

6.4.3 Discussion

The results presented in the previous section show that the inclusion of the repair operator in our EA was crucial to regularly obtain feasible solutions in the search stage. The two configurations tested show that the price to pay for an extended search is time. When using the repair operator, a special attention needs to be given to the definition of the population size and also the frequency of the usage of the operator. The results show the impact of the population size with a large difference on the average time for each iteration of the EA. Concerning the frequency of usage, in our problem, we tested the usage in larger intervals, each 50 iterations, and the results degraded. If the repair operator is used too often, the EA is unable to evolve to distinct solutions and it also seriously affects the computation time of the EA.

Preliminary tests with the EA showed that even if it can improve the population of solutions, the resulting solutions have positive infeasibility values due to the presence of duties assigned more than once and unassigned duties. The objective of setting the use of the repair operator in only a predefined number of iterations interval is to take advantage of the EA to get new (distinct) solutions, even if with some infeasibilities. After some iterations of the EA, the solutions where the crossover and mutation operator were applied are certainly very distinct from the initials and that is when the repair operator is used to “rescue” those solutions bringing them back to the space of feasible solutions. Each time a solution is repaired, the target solution of the “jump” to the feasible space depends on the duties missing and the usable empty days on the drivers’ schedules of that solution. That is how this combination of the EA and the repair operator can reach

new feasible solutions, which were unreachable with only the initial subproblem solutions included in the search space.

In the next section, the quality of the obtained solutions is evaluated and, in its discussion, final conclusions are presented, considering the aggregated information.

6.5 Solutions Evaluation

To evaluate the quality of the final solutions for the BDRP obtained with our approach, the combination of the column generation and the evolutionary algorithm with the new repair operator, we implemented the compact model for the BDRP and solve it using the branch-and-cut from CPLEX.

Table 13 includes the results from CPLEX with the time limit equal to the time used by C1, which is identified by *t1*, the time used by C2, identified by *t2* and the time limit of one hour (1h). The last columns (Δ) show the percentage distance between the best solutions found by the EA and CPLEX, concretely, the columns Δ*t1* and Δ*t2* include the differences between the best solutions of each configuration and CPLEX in the same time, and the column Δ*1h* include the difference between the best solutions of the EA (C1 or C2) and the solutions obtained by CPLEX with the time limit of one hour. The times *t1* and *t2* are, for each configuration, the sum of the time used in the column generation and the time of the EA, since our approach includes the two stages.

In Table 13, for each instance, the best integer solution value is highlighted with bold, and the underline indicates the method with the best solution in the same time (our approach vs CPLEX).

The time used by the EA search is, in average, with configuration C1 34% of the total time and with configuration C2 11% of the total time. The configuration C2 uses in the average global time, 75% of the time spent by C1.

From the observation of the results in Table 13, it is clear that CPLEX outperforms our approach in the smaller instances (p80, p100 and c122), but the inverse is true in the larger ones, which is a usual motivation for using

decomposition models. Apparently, the increase on the size of the instances causes difficulties to CPLEX, making it possible to obtain better solutions with our approach based in decomposition models and EA search.

Comparing the best solutions found by our approach and the ones from CPLEX, in the smaller instances we have a small difference for the CPLEX results, but in the larger ones, we can reach solutions with less drivers, which has a significant impact on the solution value reached, resulting that, in average, the best solutions found by our approach are 0,4% better than those found by CPLEX in one hour. In the time used by C1, our solutions are, in average, 0,61% better than CPLEX and in the time used by C2 the solutions of our algorithm have a value, in average, 0,83% lower than the solutions achieved by CPLEX.

If we focus on the comparison between the EA and CPLEX in the time used in both configurations and we also look at the three groups of instances in separate the results are distinct between groups.

In the group p80, the CPLEX reached better solutions in 9 of the 10 instances, however, in instance p80_3, in both configurations, the EA found a solution with one driver less (the difference is approximated to the fixed cost of a driver), which allows that both configurations have an average percentage difference of -0,387% in this group.

In the group of instances p100, the results of our algorithm were constantly overtaken, even if by very small percentage differences. The average absolute difference is of 205 and 226 in C1 and C2, respectively.

The group of instances c is the one where our algorithm was able to surpass CPLEX, even if it could run 1 hour. The table shows that, in this group of instances, the percentage difference of the solutions value obtained by C1 is -1,3% lower than CPLEX and that difference increases to -1,9% if the results of C2 are considered. If we concentrate on the bigger instances of the group, groups c224, c226 and c238, the differences are even more relevant, since the solution values of C1 are 1,75% lower than CPLEX and in the ones of C2 the difference raises to -2,54%.

These results show that our approach can be used to achieve good solutions for any BDRP problem and may be preferred over CPLEX when solving

larger instances of the problem, when CPLEX performance degrades, and our approach presents improved results.

Table 13 – Integer solutions comparison

Instance	Average Solution Value		Best Solution Value		Time		CPLEX Integer Solution			Δ		
	C1	C2	C1	C2	C1	C2	*t1*	*t2*	*1h*	*t1*	*t2*	*1h*
p80_1	2330017	2316645	2316543	2316564	353,1	339,1	2316261	2316270	**2316234**	0,012%	0,013%	0,013%
p80_2	1812738	1812739	1812692	1812692	157,3	150,7	1812624	1812627	**1812592**	0,004%	0,004%	0,006%
p80_3	2422591	2425963	2419156	2419149	272,6	248,0	2519068	2519068	**2418636**	-3,966%	-3,967%	0,021%
p80_4	2403635	2370114	2316440	2316454	600,0	577,8	2316157	2316158	**2316152**	0,012%	0,013%	0,012%
p80_5	2315256	2315260	2315206	2315149	400,1	358,0	2314961	2314979	**2314956**	0,011%	0,007%	0,008%
p80_6	2119243	2115913	2115818	2115836	259,1	247,9	2115671	2115657	**2115608**	0,007%	0,008%	0,010%
p80_7	2487755	2501093	2417442	2417450	543,1	518,1	2417209	2417204	**2417172**	0,010%	0,010%	0,011%
p80_8	2944538	2924451	2920965	2920980	866,3	779,9	2920507	2920507	**2920506**	0,016%	0,016%	0,016%
p80_9	2387016	2323384	2216393	2216368	600,7	547,7	**2216148***	**2216148***	**2216148***	0,011%	0,010%	0,010%
p80_10	3026759	3020124	3019911	3019911	1633,9	1356,3	**3019484**	**3019484**	**3019484**	0,014%	0,014%	0,014%
p80	2424955	2412569	23870576	2387055	568,6	512,4	2396809	2396810	2386749	-0,387%	-0,387%	0,012%
p100_1	3620953	3621007	3620855	3620881	2490,0	1834,1	**3620552***	**3620552***	**3620552***	0,008%	0,009%	0,008%
p100_2	3219823	3219839	3219732	3219763	3195,3	2863,6	**3219528**	**3219528**	**3219528**	0,006%	0,007%	0,006%
p100_3	3319235	3319281	3319154	3319177	2122,4	1446,1	**3318911**	3318917	3318914	0,007%	0,008%	0,007%
p100_4	2918208	2918232	2918149	2918118	1228,5	986,2	**2917969**	**2917969**	**2917969**	0,006%	0,005%	0,005%
p100_5	3319635	3319666	3319564	3319580	1786,9	1247,6	**3319355**	**3319355**	**3319355**	0,006%	0,007%	0,006%
p100_6	2615271	2615289	2615214	2615225	1482,2	1247,3	**2615064**	**2615064**	**2615064**	0,006%	0,006%	0,006%
p100_7	3216361	3216393	3216309	3216341	2645,2	1739,8	**3216188**	**3216188**	**3216188**	0,004%	0,005%	0,004%
p100_8	3322450	3322470	3322291	3322355	2026,7	1612,1	3321845	3321847	**3321841**	0,013%	0,015%	0,014%
p100_9	3622221	3622252	3622050	3622086	2926,4	2258,4	**3621878**	**3621878**	**3621878**	0,005%	0,006%	0,005%
p100_10	2330110	2380212	2313365	2313377	814,0	691,6	**2313348**	**2313348**	**2313348**	0,001%	0,001%	0,001%
p100	3150427	3155464	3148668	3148690	2071,8	1592,7	3148464	3148465	3148464	0,006%	0,007%	0,006%
c122_1	2217033	2217073	2216755	2216899	415,6	387,1	2216372	2216364	**2216288**	0,017%	0,024%	0,021%
c122_2	2218163	2218147	2217951	2217957	416,9	398,3	2217393	2217372	**2217211**	0,025%	0,026%	0,033%
c122_3	2229822	2226434	2219435	2219424	582,5	555,6	2218581	2218606	**2218455**	0,038%	0,037%	0,044%
c224_1	4940385	4940477	**4940156**	4940172	3031,9	1894,3	5039044	5338910	5038920	-1,962%	-7,469%	-1,960%
c224_2	5245134	5245137	5244932	5244975	3066,7	1920,5	5243969	5442753	**5243264**	0,018%	-3,634%	0,032%
c224_3	4946252	4942965	**4939380**	4939414	2809,3	1875,8	5138262	5238187	5138183	-3,871%	-5,704%	-3,869%
c226_1	4357092	4340291	**4239915**	4339894	2062,4	1833,2	4437926	4438849	4337050	-4,462%	-2,229%	-2,240%
c226_2	4468057	4468176	4440727	4440971	2233,8	1839,9	4440575	4440568	**4438216**	0,003%	0,009%	0,057%
c226_3	4532422	4532656	**4440854**	4442204	2134,9	1853,1	4540011	4539193	4539060	-2,184%	-2,137%	-2,164%
c238_1	6053308	6053492	**6052970**	6053113	3574,9	2006,8	6055270	6055709	6055270	-0,038%	-0,043%	-0,038%
c238_2	6041940	6051908	**5952222**	6051728	3133,8	1923,2	6052844	6052892	6052844	-1,662%	-0,019%	-1,662%
c238_3	5951400	5951489	**5951162**	5951287	3089,3	1899,2	6049824	6049910	6049740	-1,631%	-1,630%	-1,629%
c	4433417	4432354	4404705	4421503	2212,7	1532,3	4470839	4520776	4462042	-1,309%	-1,897%	-1,115%

6.6 Perturbations Evaluation

In this section, we describe the tests to evaluate the proposed perturbation, introduced in Section 5.4.4.

6.6.1 Configuration

The maximum available time to run each test was set to 4 hours and the maximum number of cycles to 24. A cycle iteration includes the generation of new columns followed by the metaheuristic search and the addition of new perturbations. To avoid the tail effect of the CG, each CG cycle was limited to (5 x *NumberOfSubproblems*) seconds.

Since our objective is to evaluate how the perturbations guide to an integer solution, we select the simpler metaheuristic for the search, which is the local search stopping in the first improvement.

The global solution generator used to select the initial solution for the metaheuristic search is the one that selects the subproblem solution with the highest value in the final RMP solution.

The minimum value of the variables to be considered for perturbation was set to 0,4, which means that, if all the variables corresponding to a subproblem/driver in a week have a lower value than 0,4, no perturbation is inserted for that week and driver.

6.6.2 Results

The results from the tests using perturbations are presented in Table 14.

The first column presents the total time in seconds (including the time of the initial CG with the time limit of 1800s). The next column shows the number of iterations that were run with each instance. The two columns under the label "# unassigned" present the number of unassigned duties in the roster obtained by the global solution generator in the first iteration (without any perturbation) and in the last one. The feasibility value is only presented for the instances where a solution with all the duties assigned was reached. The two columns under the label "Perturbations" show the total number of perturbations inserted and the number of perturbations added in the last iteration. The last column presents the percentage of duties from the

instance that were fixed to a driver based on the total number of perturbations and the number of duties in the instance (column Total in Table 5).

Table 14 – Results from perturbations tests

Instance	Total time	Iterations run	# unassigned		Feasibility Value	Perturbations		% duties
			First	Last		Total	Last	
p80_1	2807,0	17	61	0	2417225	441	0	97%
p80_2	1636,0	15	45	0	1913038	350	0	99%
p80_3	3590,7	24	70	3	-	454	9	96%
p80_4	2041,0	15	59	22	-	255	0	56%
p80_5	2790,3	19	54	0	2416017	440	0	99%
p80_6	2021,0	19	62	0	2216549	405	0	97%
p80_7	3883,8	23	72	0	2518390	459	0	96%
p80_8	5165,2	24	82	0	3223555	544	6	96%
p80_9	2827,7	18	58	0	2316840	424	0	96%
p80_10	5443,1	24	88	8	-	535	12	90%
p100_1	6192,6	24	114	65	-	425	34	59%
p100_2	9319,7	22	98	1	-	606	0	96%
p100_3	5223,3	24	91	24	-	487	53	75%
p100_4	4678,4	23	83	0	3219933	547	0	96%
p100_5	5215,3	24	96	18	-	541	38	83%
p100_6	3829,3	19	73	2	-	490	0	95%
p100_7	5421,6	24	100	10	-	526	46	84%
p100_8	4593,2	18	70	0	3625001	625	0	95%
p100_9	6631,8	24	100	14	-	569	56	81%
p100_10	3455,5	20	41	0	2413853	446	0	99%
c122_1	2894,9	20	49	0	2317777	425	0	99%
c122_2	3115,5	18	59	1	-	419	0	98%
c122_3	2695,5	17	62	0	2422532	412	0	96%
c224_1	8262,7	24	125	102	-	252	5	26%
c224_2	2807,0	24	142	123	-	193	13	20%
c224_3	>14400	20	111	23	-	897	24	92%
c226_1	>14400	10	143	35	-	725	67	88%
c226_2	>14400	13	140	18	-	799	21	94%
c226_3	>14400	24	147	36	-	763	52	88%
c238_1	>14400	7	234	125	-	729	154	61%
c238_2	>14400	7	231	137	-	725	156	61%
c238_3	>14400	20	215	119	-	658	50	57%

6.6.3 Discussion

The results from the tests with the use of our new perturbation show that it can be effective in the achievement of feasible solutions. For twelve instances, the ones with the feasibility value, the algorithm could reach a feasible solution. For the remaining, one of two reasons avoid that achievement. The first is the stopping criteria, the limit on the numbers of iterations was reached or the maximum time. The second reason to stop the

algorithm was the reach of an iteration where zero perturbations were inserted, which can occur when all the remaining fractional variables have a value lower than the limit defined (0,4). The instances for which the test ended because of the second reason are the ones with a zero in the column "Perturbations-Last".

The use of the proposed perturbation in the algorithm using multiple search cycles in the perturbed search spaces outperforms the results obtained by the single search in the original search space (Section 6.3). Neglecting the additional time, the algorithm using perturbations was able to obtain feasible solutions for twelve instances in an average time lower than one hour. In the results of the single search, the metaheuristic with maximum number of feasible solutions was the simulated annealing, achieving feasible solution to only seven instances.

As expected, the major drawback in the use of perturbations to solve the BDRP is the need to repeat the CG cycles, which affects very negatively the total time of the algorithm. The last seven (c224_3 to c238_3) ended in the time limit, and in two cases only seven iterations were run. This occurs because after adding each set of perturbations, the RMP is unable to obtain a feasible solution without using artificial variables and the time limit defined for each column generation cycle is only considered when a feasible solution is reached. The CG spends a large amount of time to eliminate the use of the artificial variables in the RMP by generating the needed columns considering the new constraints that are only defined in the RMP but affects the subproblem solutions to generate.

Besides the poor performance, the objective of the tests was to validate that the suggested perturbation generator could guide the CG to obtain feasible rosters, which occur in twelve instances, even if the solution found uses at least one additional driver, when comparing with the solutions from CPLEX. In the remaining instances, the improvement of the generated solution through the generations is also observed. The charts in Figure 42, Figure 43 and Figure 44 show the evolution of the number of unassigned duties (represented by lines) in the generated roster as the number of perturbations (represented by dots) increases.

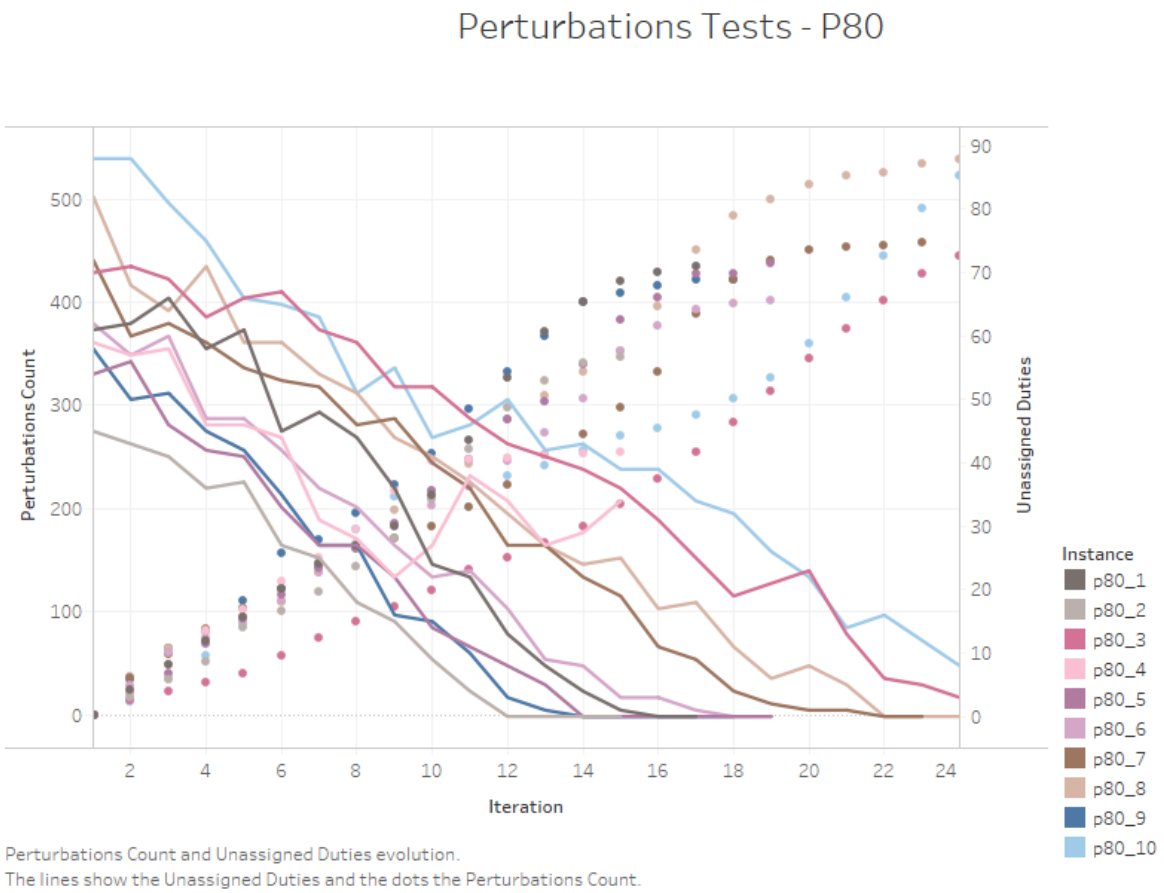

Figure 42 – Perturbations tests results - Instances P80

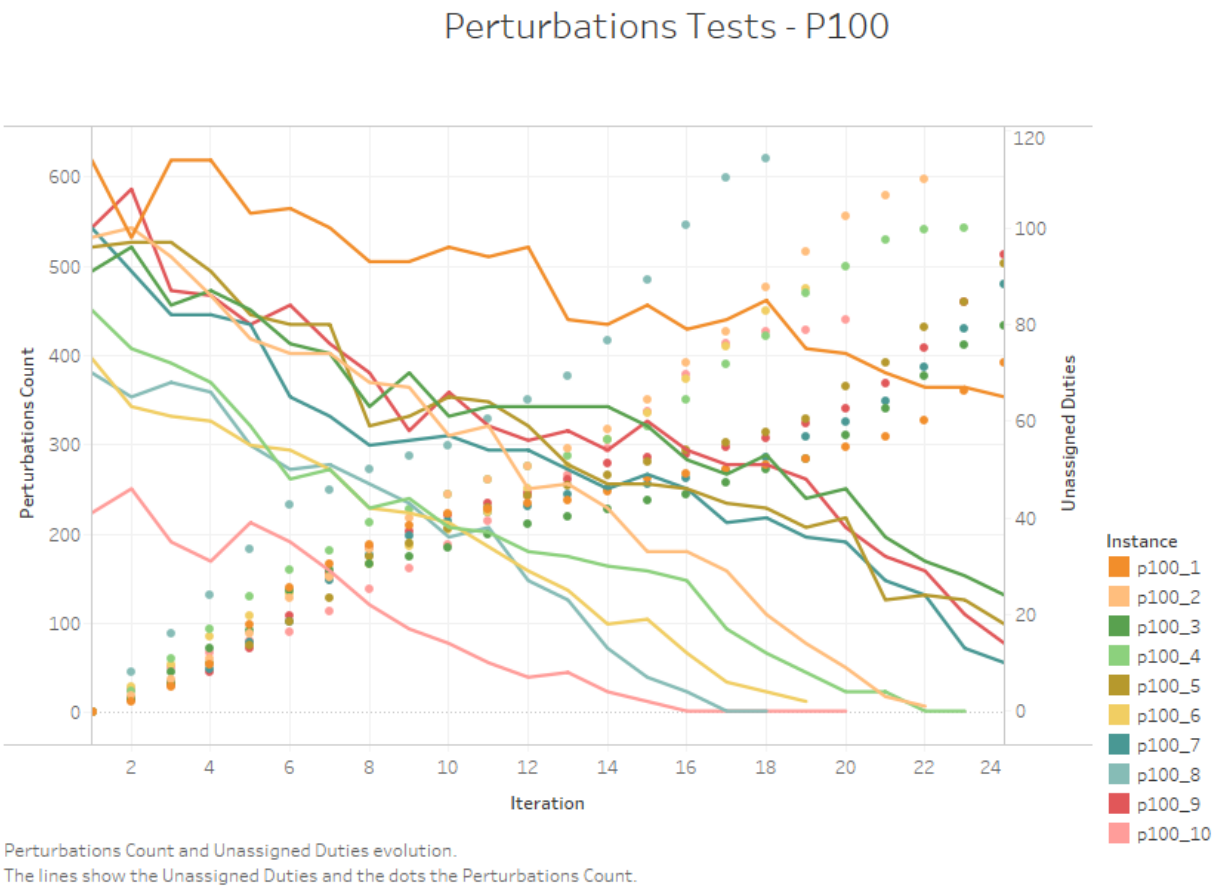

Figure 43 – Perturbations tests results - Instances P100

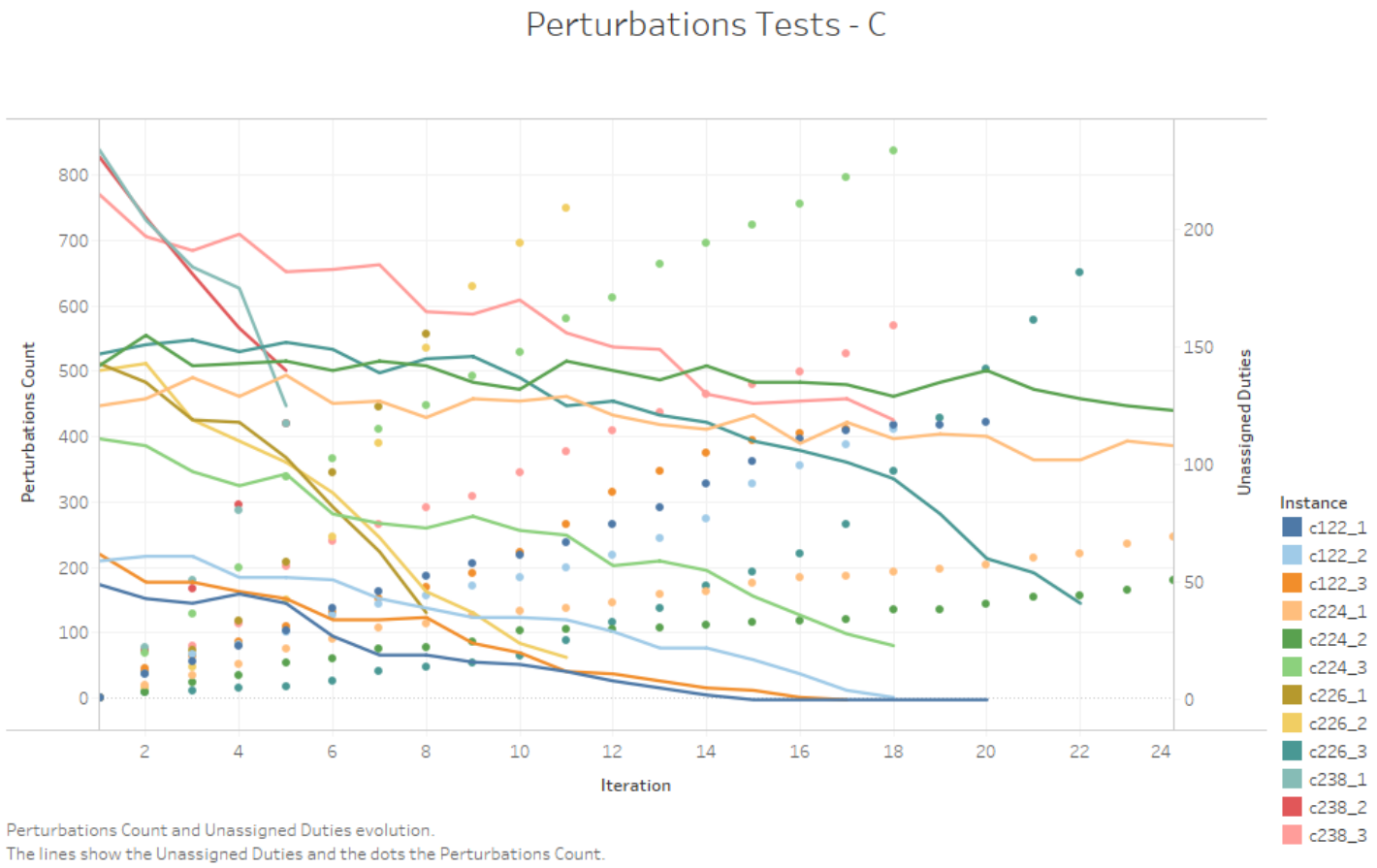

Figure 44 – Perturbations tests results - Instances C

Considering the behaviour of the CG with the decomposition model for the BDRP, these results show that the use of perturbations is not the best approach when the objective is to obtain good quality solutions in a short time. The results also allow to reinforce the advantage of using the proposed repair operator as a generator of missing subproblem solutions.

In the group of instances p80 (Figure 42), for most of the instances a feasible solution was achieved (zero duties not assigned). With two instances the algorithm reached the limit of iterations and in one has stopped in the iteration 15 with around 30 duties not assigned.

In the group of instances p100 (Figure 43), for five instances the algorithm reached the limit of iterations with a positive number of unassigned duties and for the instances p100_2 and p100_6, the algorithm stopped with two and one duties not assigned, respectively.

In the group of instances c (Figure 44), only for the smaller instances (c122) the method was able to obtain feasible solutions. In the bigger instances, the method reached the time limit or the limit of iterations.

6.7 Summary

This chapter presented computational tests run during our research using a set of 32 instances from the literature. The first set of tests evaluated some alternative configurations on the column generation and using the decomposition models (with the three distinct subproblem representations). From the first set of tests, the search space of a selected configuration was used to run a new set of tests to compare our basic evolutionary algorithm (without using the repair operator) with some single-solution metaheuristics.

Our evolutionary algorithm was tested using a search space obtained with the best configuration from the first set of tests, but with a smaller time limit, using two distinct configurations to evaluate the difference on the computational time. The quality of the results obtained was assessed by comparing our results with the results obtained solving the compact model of the problem with a commercial solver.

A final set of tests was run to evaluate if the use of perturbations, particularly with our generator, is effective in reaching feasible solutions.

7 Conclusions and Future Work

This thesis describes the research on new hybrid methods to address a hard combinatorial optimization problem. The addressed problem is the Bus Driver Rostering Problem, a problem belonging to the wider class of Personnel Scheduling or Rostering Problems to which the research stills very active in the literature.

The proposed approach to the problem, classified as a hybrid method, involves primarily the combination of an exact method, the column generation, with a metaheuristic, an evolutionary algorithm. However, other hybridizations occur in the described research as multiple metaheuristics were used in combination with the column generation and even inside the column generation where exact methods and heuristics co-exist to obtain solutions to the subproblems.

7.1 Conclusions

The essential element to the development of the proposed research was the definition of the decomposition model to represent the problem and allow the use of the column generation. New methods to obtain subproblem solutions in column generation were developed and new variants were evaluated. Two additional subproblem models were also proposed and tested. The results of the research pointed the decomposition model with the subproblem represented by the assignment model solved by the column generation using the roster heuristic while it reaches feasible solutions as the best configuration for the first stage of the proposed approach.

A basic version of the developed EA was compared with single-solution metaheuristics to compare its ability to search for complete integer solutions in the search space obtained by the column generation. Results of the exploration of the search space with the metaheuristics revealed that all the metaheuristics were able to achieve improved solutions (when comparing with the best initial solution), with a higher accuracy from the simulated annealing metaheuristic, but all were unable to achieve feasible solutions

with all the duties assigned, which determines the need to include additional subproblem solutions in the search space.

To avoid the time needed to run multiple cycles of column generation, the repair operator was proposed and included in the final EA as an alternative to generate new subproblem solutions which result from global solutions repaired (removing duplicated assignments and adding missing duties).

The development of the repair operator was critical for the capacity of the EA in reaching feasible solutions.

The final algorithm was capable to regularly achieve feasible solutions for all the instances tested. The quality of those solutions is assessed by the comparison with the results achieved by a commercial solver used to solve the instances represented by a compact model of the problem. The use of perturbations and multiple cycles of column generations was tested but show that it is not an efficient alternative to solve the problem in short time.

Our approach is an alternative to solve the generality of the BDRP instances to obtain good solutions, but we revealed that the proposed approach can outperform a commercial solver by achieving better results in some larger instances.

We now return to the objectives announced in the introduction to describe how they were fulfilled:

"Define a decomposition model for the bus driver rostering problem solvable by using the column generation method;"

The definition of a decomposition model was part of the initial research since it is a base element to the proposed approach. Besides the decomposition model resulting from applying the Dantzig-Wolfe decomposition to the compact model, two other reformulations were proposed in this research.

"Obtain a good quality search space with a sufficient quantity and diversity of partial solutions resulting from the use of the column generation method;"

The computational tests in the column generation stage show fluctuations on the number of partial solutions (subproblem solutions), especially due to the number of iterations of the CG, which is largely impacted by the use, or not, of the heuristics as subproblem solvers. The characteristics of the problem

in study result in optimal linear solutions very fractional, which led us to develop the roster heuristic, to obtain complementary solutions, and the repair operator, to include new solutions in the search space. With these two components, the search space includes the partial solutions that allow to achieve good quality rosters.

"Define a new evolutionary algorithm adapted to the solutions representation based on partial solutions which are subproblem solutions obtained by the column generation;"

The new evolutionary algorithm was developed and improved to address the BDRP, however, as described in Section 4.3, all the main elements of the algorithm were designed to be independent of the problem or, on the other hand, allow each problem to tailor the desired operators by implementing the methods inside the decomposition model and not in the metaheuristic.

"Develop the complete hybrid algorithm combining the column generation stage and the metaheuristic search with the necessary configuration options allowing the user to define all the parameters and components to be used;"

The complete hybrid algorithm is described in Chapter 5. All the options in the definition of the search space and in the metaheuristic search are described. For the BDRP, one can select one of three decompositions, two heuristics to solve the subproblems, five metaheuristics to explore the search space, decide to use the repair operator or the perturbations, etc. In the majority of these components, important parameters can be set in runtime, as the selection of global solutions generators, population size used by the EA, crossover and mutation percentage, etc.

"Provide a skeleton of the algorithm components allowing the development of new combinations with distinct metaheuristics, particularly, other population-based metaheuristics;"

As described in Section 4.2, all the developments are prepared to the future inclusion of new metaheuristics in the framework which will share the operators or components developed. Besides the inclusion of new components, or alternatives inside the components, a diversity of choices already exists, as described in Section 5.6, which allow the generations of multiple variants of a global algorithm to address a problem represented by a decomposition model.

"Obtain good quality solutions for the BDRP instances in study;"

The results of the computational tests, particularly the tests of the EA with the repair operator, show that the algorithm reaches regularly good solutions for the tests instances of the problem.

"Reach an algorithm configuration which can be competitive in comparison with other existing methods to solve the problem in study."

We found the problem and the conditions where our proposed approach performs better than the branch-and-cut implementation of a commercial solver. The results show that in the smaller instances we can reach good results, however, the solutions from CPLEX are better. In the larger instances, the performance of the CPLEX decreases and our approach obtains the best results.

Having listed the objectives and explaining how we consider they were fulfilled, we will now make use of our research to answer the research question presented in the introduction.

Is the hybridization of column generation method with an evolutionary algorithm effective for attaining good quality solutions for rostering problems in reasonable time?

Our research question inquires if the proposed approach, the hybridization of column generation method with an evolutionary algorithm, can be an alternative method to solve the class of problems in study, the rostering problems. Even before the development and tests of our approach, the review of the literature made it clear that there is no method capable to solve all the rostering problems. The diversity of approaches found in the literature and the amount of publications about the problem by the same groups of researchers show that the search for the best algorithm for each particular problem stills active in the research community, with each researcher, or group, testing multiple approaches on the same problem or testing the same method in distinct problems.

Our approach intents to be easily expandable to a diversity of problems, not only the rostering problems, but also other combinatorial optimization problems, if they can be reformulated by a decomposition model. As occurs with the SearchCol framework, the usage of our method in a distinct problem does not need significant work, at least, for the standard behaviour, without consideration on problem particularities.

For the BDRP, the developed algorithm is *effective for attaining good quality solutions*, as was shown by the computational tests. However, to

reach these results *in reasonable time*, significant improvements in the algorithm and knowledge about the problem were needed. When addressing a new problem, the behaviour of the algorithm may be better or worse, but we expect that the improvements we describe in this thesis, if needed, will have the same impact as they did with the BDRP.

7.2 Future work

During the research described in this thesis, multiple new research paths emerged, some due to difficulties from the problem in study, others from new ideas or third-party suggestions (reviewers, conferences, PhD seminars, etc.). Some of those paths were included in the research, some were only briefly explored due to time constraints, and others were automatically postponed to future work.

Consequently, as future work, we intend to:

- Add stabilization methods to the column generation to try to reduce the time spent on it;
- Deeply explore the use of Constraint Programming to address the BDRP, not only to solve the subproblem, obtaining more than one solution in each iteration, but also the entire compact problem;
- Extend the research to address the objective of reducing the inequity between drivers, which was included in the implementation of the roster heuristic but not fully explored;
- Relax the constraints related with the previous rostering periods to allow that the generated work-schedules can be swapped between drivers. Evaluate the impact on the global performance of the algorithm and the required work to adapt the solution to the original problem.
- Explore the definition of a new decomposition structure (ex.: by week), to have, in the search space, smaller solutions to combine, increasing the capacity of the metaheuristics to avoid the over-assignments and/or under-assignments;
- Define templates for the days-off or predefined lines of work to assign to the drivers;

- Test our approach in other rostering or general combinatorial optimization problems.

With the finish of this thesis, new research opportunities arise. We hope to continue to contribute to this exciting area of solving complex decision problems.

References

Aarts, E., & Lenstra, J. K. (1997). *Local Search in Combinatorial Optimization*: John Wiley & Sons, Inc.

Ahuja, R. K., Magnanti, T. L., & Orlin, J. B. (1993). *Network flows: theory, algorithms, and applications*: Prentice-Hall, Inc.

Aickelin, U., & Dowsland, K. (2000). Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem. *Journal of Scheduling*(3), 139--153.

Aickelin, U., & Dowsland, K. A. (2004). An indirect genetic algorithm for a nurse-scheduling problem. *Comput. Oper. Res.*, *31*(5), 761-778. doi:10.1016/s0305-0548(03)00034-0

Aickelin, U., & White, P. (2004). Building Better Nurse Scheduling Algorithms. *Annals of Operations Research, 128*(1), 159-177. doi:10.1023/B:ANOR.0000019103.31340.a6

Alfares, H. K. (1998). An efficient two-phase algorithm for cyclic days-off scheduling. *Computers & Operations Research, 25*(11), 913-923. doi:Doi: 10.1016/s0305-0548(98)00033-1

Alfares, H. K. (2006). Compressed workweek scheduling with days-off consecutivity, weekend-off frequency, and work stretch constraints. *INFOR, 44*(3), 175-189.

Alfieri, A., Kroon, L., & Van De Velde, S. (2007). Personnel scheduling in a complex logistic system: a railway application case. *Journal of Intelligent Manufacturing, 18*(2), 223-232. doi:10.1007/s10845-007-0017-9

Alvelos, F., de Sousa, A., & Santos, D. (2010). SearchCol: Metaheuristic Search by Column Generation. In M. Blesa, C. Blum, G. Raidl, A. Roli, & M. Sampels (Eds.), *Hybrid Metaheuristics* (Vol. 6373, pp. 190-205): Springer, Berlin, Heidelberg.

Alvelos, F., Silva, E., & de Carvalho, J. (2014). A Hybrid Heuristic Based on Column Generation for Two- and Three- Stage Bin Packing Problems. In B. Murgante, S. Misra, A. C. Rocha, C. Torre, J. Rocha, M. Falcão, D. Taniar, B. Apduhan, & O. Gervasi (Eds.), *Computational Science and Its Applications – ICCSA 2014* (Vol. 8580, pp. 211-226): Springer International Publishing.

Alvelos, F., Sousa, A., & Santos, D. (2013). Combining column generation and metaheuristics. In E.-G. Talbi (Ed.), *Hybrid metaheuristics* (Vol. 434, pp. 285-334): Springer.

Ásgeirsson, E. (2014). Bridging the gap between self schedules and feasible schedules in staff scheduling. *Annals of Operations Research, 218*(1), 51-69. doi:10.1007/s10479-012-1060-2

Awadallah, M. A., Bolaji, A. L. a., & Al-Betar, M. A. (2015). A hybrid artificial bee colony for a nurse rostering problem. *Applied Soft Computing, 35*, 726-739. doi:http://dx.doi.org/10.1016/j.asoc.2015.07.004

Bai, R., Burke, E. K., Kendall, G., Jingpeng, L., & McCollum, B. (2010). A Hybrid Evolutionary Approach to the Nurse Rostering Problem. *Evolutionary Computation, IEEE Transactions on, 14*(4), 580-590. doi:10.1109/tevc.2009.2033583

Baker, K. R. (1974). Scheduling a Full-Time Workforce to Meet Cyclic Staffing Requirements. *Management Science, 20*(12), 1561-1568. doi:10.2307/2630177

Bard, J. F., & Purnomo, H. W. (2004). Preference scheduling for nurses using column generation. *European Journal of Operational Research, 164*(2), 510-534.

Bard, J. F., & Purnomo, H. W. (2005). Short-term nurse scheduling in response to daily fluctuations in supply and demand. *Health Care Management Science, 8*(4), 315-324. doi:10.1007/s10729-005-4141-9

Beliën, J., & Demeulemeester, E. (2007). On the trade-off between staff-decomposed and activity-decomposed column generation for a staff scheduling problem. *Annals of Operations Research, 155*(1), 143-166. doi:10.1007/s10479-007-0220-2

Bilgin, B., De Causmaecker, P., & Vanden Berghe, G. (2009). *A hyperheuristic approach to belgian nurse rostering problems.* Paper presented at the Proceedings of the 4th Multidisciplinary International Conference on Scheduling: Theory and Applications.

Bilgin, B., Demeester, P., Misir, M., Vancroonenburg, W., & Vanden Berghe, G. (2012). One hyper-heuristic approach to two timetabling problems in health care. *Journal of Heuristics, 18*(3), 401-434. doi:10.1007/s10732-011-9192-0

Blum, C., Puchinger, J., Raidl, G. R., & Roli, A. (2011). Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing, 11*(6), 4135-4151. doi:http://dx.doi.org/10.1016/j.asoc.2011.02.032

Borndörfer, R., Reuther, M., Schlechte, T., Schulz, C., Swarat, E., & Weider, S. (2015). *Duty Rostering in Public Transport - Facing Preferences, Fairness, and Fatigue*. Retrieved from

Borndörfer, R., Schulz, C., Seidl, S., & Weider, S. (2017). Integration of duty scheduling and rostering to increase driver satisfaction. *Public Transport, 9*(1), 177-191. doi:10.1007/s12469-017-0153-3

Bourdais, S., Galinier, P., & Pesant, G. (2003). *HIBISCUS: A constraint programming application to staff scheduling in health care.* Paper

presented at the Principles and Practice of Constraint Programming–CP 2003.

Brucker, P., Burke, E., Curtois, T., Qu, R., & Vanden Berghe, G. (2010). A shift sequence based approach for nurse scheduling and a new benchmark dataset. *Journal of Heuristics, 16*(4), 559-573. doi:10.1007/s10732-008-9099-6

Brunner, J., & Edenharter, G. (2011). Long term staff scheduling of physicians with different experience levels in hospitals using column generation. *Health Care Management Science, 14*(2), 189-202. doi:10.1007/s10729-011-9155-x

Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P., & Schulenburg, S. (2003a). Hyper-Heuristics: An Emerging Direction in Modern Search Technology. In F. Glover & G. Kochenberger (Eds.), *Handbook of Metaheuristics* (Vol. 57, pp. 457-474): Springer US.

Burke, E. K., Curtois, T., Post, G., Qu, R., & Veltman, B. (2008). A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem. *European Journal of Operational Research, 188*(2), 330-341. doi:http://dx.doi.org/10.1016/j.ejor.2007.04.030

Burke, E. K., De Causmaecker, P., Berghe, G., & Van Landeghem, H. (2004). The State of the Art of Nurse Rostering. *Journal of Scheduling, 7*(6), 441-499. doi:10.1023/B:JOSH.0000046076.75950.0b

Burke, E. K., Kendall, G., & Soubeiga, E. (2003b). A Tabu-Search Hyperheuristic for Timetabling and Rostering. *Journal of Heuristics, 9*(6), 451-470. doi:10.1023/B:HEUR.0000012446.94732.b6

Burke, E. K., Li, J., & Qu, R. (2010). A hybrid model of integer programming and variable neighbourhood search for highly-constrained nurse rostering problems. *European Journal of Operational Research, 203*(2), 484-493. doi:http://dx.doi.org/10.1016/j.ejor.2009.07.036

Cappanera, P., & Gallo, G. (2004). A Multicommodity Flow Approach to the Crew Rostering Problem. *Operations Research, 52*(4), 583-596. doi:doi:10.1287/opre.1040.0110

Caprara, A., Toth, P., Vigo, D., & Fischetti, M. (1998). Modeling and Solving the Crew Rostering Problem. *Operations Research, 46*(6), 820-830. doi:doi:10.1287/opre.46.6.820

Carter, M. W., & Lapierre, S. D. (2001). Scheduling Emergency Room Physicians. *Health Care Management Science, 4*, 347-360.

Cheng, B. M. W., Lee, J. H. M., & Wu, J. C. K. (1997). A nurse rostering system using constraint programming and redundant modeling. *IEEE Transactions on Information Technology in Biomedicine, 1*(1), 44-54. doi:10.1109/4233.594027

Chiaramonte, M., & Caswell, D. (2016). Rerostering of nurses with intelligent agents and iterated local search. *IIE Transactions on Healthcare Systems Engineering, 6*(4), 213-222. doi:10.1080/19488300.2016.1226211

Chiaramonte, M., Cochran, J., & Caswell, D. (2015). Nurse preference rostering using agents and iterated local search. *Annals of Operations Research, 226*(1), 443-461. doi:10.1007/s10479-014-1701-8

Chiaramonte, M. V., & Chiaramonte, L. M. (2008). An agent-based nurse rostering system under minimal staffing conditions. *International Journal of Production Economics, 114*, 697-713. doi:10.1016/j.ijpe.2008.03.004

Cintra, G., & Wakabayashi, Y. (2004). Dynamic Programming and Column Generation Based Approaches for Two-Dimensional Guillotine Cutting Problems. In C. Ribeiro & S. Martins (Eds.), *Experimental and Efficient Algorithms* (Vol. 3059, pp. 175-190): Springer Berlin Heidelberg.

Cipriano, R., Di Gaspero, L., & Dovier, A. (2006). Hybrid Approaches for Rostering: A Case Study in the Integration of Constraint Programming and Local Search. In F. Almeida, M. Blesa Aguilera, C. Blum, J. Moreno Vega, M. Pérez Pérez, A. Roli, & M. Sampels (Eds.), *Hybrid Metaheuristics* (Vol. 4030, pp. 110-123): Springer Berlin / Heidelberg.

Costa, M.-C., Jarray, F., & Picouleau, C. (2006). An acyclic days-off scheduling problem. *4OR: A Quarterly Journal of Operations Research, 4*(1), 73-85. doi:10.1007/s10288-005-0086-6

Dantzig, G. B., & Wolfe, P. (1960). Decomposition Principle for Linear Programs. *Operations Research, 8*(1), 101-111.

De Bruecker, P., Van den Bergh, J., Beliën, J., & Demeulemeester, E. (2015). Workforce planning incorporating skills: State of the art. *European Journal of Operational Research, 243*(1), 1-16. doi:http://dx.doi.org/10.1016/j.ejor.2014.10.038

De Causmaecker, P., Demeester, P., Berghe, G. V., & Verbeke, B. (2005). *A coordination model for distributed personnel scheduling.* Paper presented at the Proceedings of the 19 th Orbel Conference, Louvain-La-Neuve.

de Matta, R., & Peters, E. (2009). Developing work schedules for an inter-city transit system with multiple driver types and fleet types. *European Journal of Operational Research, 192*(3), 852-865. doi:http://dx.doi.org/10.1016/j.ejor.2007.09.045

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation, 6*(2), 182-197. doi:10.1109/4235.996017

Desaulniers, G., Desrosiers, J., & Solomon, M. M. (2005). *Column Generation*. New York: Springer.

Dorigo, M., Birattari, M., & Stutzle, T. (2006). Ant colony optimization. *Computational Intelligence Magazine, IEEE, 1*(4), 28-39. doi:10.1109/mci.2006.329691

Dorne, R. (2008). Personnel Shift Scheduling and Rostering. In C. Voudouris, D. Lesaint, & G. Owusu (Eds.), *Service Chain Management* (pp. 125-138): Springer Berlin Heidelberg.

dos Santos, A. G., & Mateus, G. R. (2009). *General hybrid column generation algorithm for crew scheduling problems using genetic algorithm*. Paper presented at the Evolutionary Computation, 2009. CEC '09. IEEE Congress on.

Dowsland, K. A. (1993). Simulated annealing. In R. R. Colin (Ed.), *Modern heuristic techniques for combinatorial problems* (pp. 20-69): John Wiley & Sons, Inc.

Dumitrescu, I., & Stützle, T. (2003). Combinations of Local Search and Exact Algorithms. In S. Cagnoni, C. Johnson, J. R. Cardalda, E. Marchiori, D. Corne, J.-A. Meyer, J. Gottlieb, M. Middendorf, A. Guillot, G. Raidl, & E. Hart (Eds.), *Applications of Evolutionary Computing* (Vol. 2611, pp. 211-223): Springer Berlin Heidelberg.

Dumitrescu, I., & Stützle, T. (2010). Usage of Exact Algorithms to Enhance Stochastic Local Search Algorithms. In V. Maniezzo, T. Stützle, & S. Voß (Eds.), *Matheuristics* (Vol. 10, pp. 103-134): Springer US.

Elshafei, M., & Alfares, H. (2008). A dynamic programming algorithm for days-off scheduling with sequence dependent labor costs. *Journal of Scheduling, 11*(2), 85-93. doi:10.1007/s10951-007-0040-x

Ernst, A. T., Jiang, H., Krishnamoorthy, M., Owens, B., & Sier, D. (2004a). An Annotated Bibliography of Personnel Scheduling and Rostering. *Annals of Operations Research, 127*(1), 21-144. doi:10.1023/B:ANOR.0000019087.46656.e2

Ernst, A. T., Jiang, H., Krishnamoorthy, M., & Sier, D. (2004b). Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research, 153*(1), 3-27.

Fahle, T., Junker, U., Karisch, S. E., Kohl, N., Sellmann, M., & Vaaben, B. (2002). Constraint Programming Based Column Generation for Crew Assignment. *Journal of Heuristics, 8*(1), 59-81. doi:10.1023/a:1013613701606

Feo, T., & Resende, M. C. (1995). Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization, 6*(2), 109-133. doi:10.1007/bf01096763

Florêncio, L., Pimentel, C., & Alvelos, F. (2015). An Exact and a Hybrid Approach for a Machine Scheduling Problem with Job Splitting. In J. P. Almeida, J. F. Oliveira, & A. A. Pinto (Eds.), *Operational Research* (Vol. 4, pp. 191-212): Springer International Publishing.

Frey, L., Hanne, T., & Dornberger, R. (2009, 18-21 May 2009). *Optimizing staff rosters for emergency shifts for doctors.* Paper presented at the Evolutionary Computation, 2009. CEC '09. IEEE Congress on.

Fügener, A., Brunner, J. O., & Podtschaske, A. (2015). Duty and workstation rostering considering preferences and fairness: a case study at a department of anaesthesiology. *International Journal of Production Research*, 1-23. doi:10.1080/00207543.2015.1082667

Gendreau, M., Ferland, J., Gendron, B., Hail, N., Jaumard, B., Lapierre, S., . . . Soriano, P. (2007). *Physician scheduling in emergency rooms*. Paper presented at the Proceedings of the 6th international conference on Practice and theory of automated timetabling VI, Brno, Czech Republic.

Glover, F., & Kochenberger, G. (2003). *Handbook of Metaheuristics (International Series in Operations Research & Management Science)*: Springer.

Glover, F., & Laguna, M. (1993). Tabu search. In R. R. Colin (Ed.), *Modern heuristic techniques for combinatorial problems* (pp. 70-150): John Wiley & Sons, Inc.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*: Addison-Wesley Longman Publishing Co., Inc.

Goodman, M., Dowsland, K., & Thompson, J. (2009). A grasp-knapsack hybrid for a nurse-scheduling problem. *Journal of Heuristics*, *15*(4), 351-379. doi:10.1007/s10732-007-9066-7

Gröbner, M., & Wilke, P. (2001). Optimizing Employee Schedules by a Hybrid Genetic Algorithm. In E. Boers (Ed.), *Applications of Evolutionary Computing* (Vol. 2037, pp. 463-472): Springer Berlin / Heidelberg.

Günther, M., & Nissen, V. (2010). Particle Swarm Optimization and an Agent-Based Algorithm for a Problem of Staff Scheduling. In C. Di Chio, A. Brabazon, G. Di Caro, M. Ebner, M. Farooq, A. Fink, J. Grahl, G. Greenfield, P. Machado, M. O'Neill, E. Tarantino, & N. Urquhart (Eds.), *Applications of Evolutionary Computation* (Vol. 6025, pp. 451-461): Springer Berlin / Heidelberg.

Hanafi, R., & Kozan, E. (2014). A hybrid constructive heuristic and simulated annealing for railway crew scheduling. *Computers and Industrial Engineering*, *70*(1), 11-19. doi:10.1016/j.cie.2014.01.002

Hartog, A., Huisman, D., Abbink, E. J. W., & Kroon, L. G. (2009). Decision support for crew rostering at NS. *Public Transport*, *1*(2), 121-133. doi:10.1007/s12469-009-0009-6

Holland, J. H. (1992). *Adaptation in natural and artificial systems*: MIT Press.

Ibarra-Rojas, O. J., Delgado, F., Giesen, R., & Muñoz, J. C. (2015). Planning, operation, and control of bus transport systems: A literature review. *Transportation Research Part B: Methodological*, *77*, 38-75. doi:http://dx.doi.org/10.1016/j.trb.2015.03.002

IBM. (2016a). CPLEX CP Optimizer. Retrieved from https://www-01.ibm.com/software/commerce/optimization/cplex-cp-optimizer/

IBM. (2016b). CPLEX Optimizer. Retrieved from http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/index.html

Johnson, D. S., Papadimitriou, C. H., & Yannakakis, M. (1988). How easy is local search? *Journal of Computer and System Sciences*, *37*(1), 79-100. doi:http://dx.doi.org/10.1016/0022-0000(88)90046-3

Jütte, S., & Thonemann, U. W. (2012). Divide-and-price: A decomposition algorithm for solving large railway crew scheduling problems. *European*

Journal of Operational Research, 219(2), 214-223. doi:http://dx.doi.org/10.1016/j.ejor.2011.12.038

Kaplansky, E., & Meisels, A. (2007). Distributed personnel scheduling—negotiation among scheduling agents. *Annals of Operations Research, 155*(1), 227-255. doi:10.1007/s10479-007-0206-0

Kennedy, J., & Eberhart, R. (1995, Nov/Dec 1995). *Particle swarm optimization.* Paper presented at the Neural Networks, 1995. Proceedings., IEEE International Conference on.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science, 220*(4598), 671-680. doi:10.1126/science.220.4598.671

Kohl, N., & Karisch, S. E. (2004). Airline Crew Rostering: Problem Types, Modeling, and Optimization. *Annals of Operations Research, 127*(1), 223-257. doi:10.1023/B:ANOR.0000019091.54417.ca

Kuhn, H. W. (1955). The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly, 2*(1-2), 83-97. doi:10.1002/nav.3800020109

Kuo, Y.-H., Leung, J. M. Y., & Yano, C. A. (2014). Scheduling of Multi-skilled Staff Across Multiple Locations. *Production and Operations Management, 23*(4), 626-644. doi:10.1111/poms.12184

Kyngas, J., & Nurmi, K. (2011). Days-off scheduling for a bus transportation company. *Int. J. Innov. Comput. Appl., 3*(1), 42-49. doi:10.1504/ijica.2011.037950

Kyngas, N., Kyngas, J., & Nurmi, K. (2012). Optimizing Large-Scale Staff Rostering Instances. *Lecture Notes in Engineering and Computer Science, 2196*(1), 1524-1531.

Lagatie, R., Haspeslagh, S., & De Causmaecker, P. (2009, 29-30 October 2009). *Negotiation Protocols for Distributed Nurse Rostering.* Paper presented at the Proceedings of the 21st Benelux Conference on Artificial Intelligence, Eindhoven, Netherlands.

Law, Y. C., Lee, J. H. M., Walsh, T., & Yip, J. Y. K. (2007). Breaking Symmetry of Interchangeable Variables and Values. In C. Bessière (Ed.), *Principles and Practice of Constraint Programming – CP 2007* (Vol. 4741, pp. 423-437): Springer Berlin Heidelberg.

Leone, R., Festa, P., & Marchitto, E. (2011). A Bus Driver Scheduling Problem: a new mathematical model and a GRASP approximate solution. *Journal of Heuristics, 17*(4), 441-466. doi:10.1007/s10732-010-9141-3

Lezaun, M., Pérez, G., & Sáinz de la Maza, E. (2007). Rostering in a rail passenger carrier. *Journal of Scheduling, 10*(4), 245-254. doi:10.1007/s10951-007-0024-x

Li, H., Lim, A., & Rodrigues, B. (2003). *A hybrid AI approach for nurse rostering problem.* Paper presented at the Proceedings of the 2003 ACM symposium on Applied computing.

Lourenço, H., Martin, O., & Stützle, T. (2003). Iterated Local Search. In F. Glover & G. Kochenberger (Eds.), *Handbook of Metaheuristics* (Vol. 57, pp. 320-353): Springer US.

Lübbecke, M. E., & Desrosiers, J. (2005). Selected Topics in Column Generation. *Oper. Res., 53*(6), 1007-1023. doi:10.1287/opre.1050.0234

Lučic, P., & Teodorovic, D. (1999). Simulated annealing for the multi-objective aircrew rostering problem. *Transportation Research Part A: Policy and Practice, 33*(1), 19-45. doi:http://dx.doi.org/10.1016/S0965-8564(98)00021-4

Lučić, P., & Teodorović, D. (2007). Metaheuristics approach to the aircrew rostering problem. *Annals of Operations Research, 155*(1), 311-338. doi:10.1007/s10479-007-0216-y

Maenhout, B., & Vanhoucke, M. (2008). Comparison and hybridization of crossover operators for the nurse scheduling problem. *Annals of Operations Research, 159*(1), 333-353. doi:10.1007/s10479-007-0268-z

Maenhout, B., & Vanhoucke, M. (2010). Branching strategies in a branch-and-price approach for a multiple objective nurse scheduling problem. *Journal of Scheduling, 13*(1), 77-93. doi:10.1007/s10951-009-0108-x

Martin, S., Ouelhadj, D., Smet, P., Vanden Berghe, G., & Özcan, E. (2013). Cooperative search for fair nurse rosters. *Expert Systems with Applications, 40*(16), 6674-6683. doi:http://dx.doi.org/10.1016/j.eswa.2013.06.019

Martins, I., Alvelos, F., & Constantino, M. (2015). Decompositions and a Matheuristic for a Forest Harvest Scheduling Problem. In J. P. Almeida, J. F. Oliveira, & A. A. Pinto (Eds.), *Operational Research* (Vol. 4, pp. 237-260): Springer International Publishing.

Martins, L. d. C., & Silva, G. P. (2016, 1-4 Nov. 2016). *A Genetic Algorithm for the mass transit crew rostering problem.* Paper presented at the 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC).

Mehrotra, A., Murphy, K. E., & Trick, M. A. (2000). Optimal shift scheduling: a branch-and-price approach. *Naval Research Logistics, 47*, 185-200.

Mesquita, M., Moz, M., Paias, A., & Pato, M. (2015). A decompose-and-fix heuristic based on multi-commodity flow models for driver rostering with days-off pattern. *European Journal of Operational Research, 245*(2), 423-437. doi:http://dx.doi.org/10.1016/j.ejor.2015.03.030

Michalewicz, Z., & Schoenauer, M. (2003). Evolutionary Algorithms. In B. Editor-in-Chief: Hossein (Ed.), *Encyclopedia of Information Systems* (pp. 259-267). New York: Elsevier.

Mitchell, M. (1996). *An introduction to genetic algorithms*: MIT Press.

Mladenović, N., & Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research, 24*(11), 1097-1100. doi:http://dx.doi.org/10.1016/S0305-0548(97)00031-2

Monfroglio, A. (1996). Hybrid Genetic Algorithms for a Rostering Problem. *Software: Practice and Experience, 26*(7), 851-862. doi:10.1002/(sici)1097-024x(199607)26:7<851::aid-spe38>3.0.co;2-a

Moz, M., & Pato, M. (2003). An Integer Multicommodity Flow Model Applied to the Rerostering of Nurse Schedules. *Annals of Operations Research, 119*(1-4), 285-301. doi:10.1023/a:1022907212477

Moz, M., & Pato, M. (2007). A genetic algorithm approach to a nurse rerostering problem. *Computers & Operations Research, 34*(3), 667-691. doi:10.1016/j.cor.2005.03.019

Moz, M., Respício, A., & Pato, M. (2009). Bi-objective evolutionary heuristics for bus driver rostering. *Public Transport, 1*(3), 189-210.

Naudin, É., Chan, P. Y. C., Hiroux, M., Zemmouri, T., & Weil, G. (2012). Analysis of three mathematical models of the Staff Rostering Problem. *Journal of Scheduling, 15*(1), 23-38. doi:10.1007/s10951-009-0155-3

Nishi, T., Sugiyama, T., & Inuiguchi, M. (2014). Two-level decomposition algorithm for crew rostering problems with fair working condition. *European Journal of Operational Research, 237*(2), 465-473. doi:http://dx.doi.org/10.1016/j.ejor.2014.02.010

Nurmi, K., Kyngäs, J., & Post, G. (2011). Driver rostering for bus transit companies. *Engineering Letters, 19*(2), 125-132.

Ouelhadj, D., Martin, S., Smet, P., Ozcan, E., & Vanden Berghe, G. (2012). *Fairness in nurse rostering*. Retrieved from http://eprints.port.ac.uk/6439/

Özcan, E. (2005). Memetic Algorithms for Nurse Rostering. In P. Yolum, T. Güngör, F. Gürgen, & C. Özturan (Eds.), *Computer and Information Sciences - ISCIS 2005* (Vol. 3733, pp. 482-492): Springer Berlin / Heidelberg.

Özcan, E. (2007). Memes, Self-generation and Nurse Rostering. In E. Burke & H. Rudová (Eds.), *Practice and Theory of Automated Timetabling VI* (Vol. 3867, pp. 85-104): Springer Berlin / Heidelberg.

Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*: Addison-Wesley.

Peng, K., Shen, Y., & Li, J. (2015). A Multi-objective Simulated Annealing for Bus Driver Rostering. In M. Gong, L. Pan, T. Song, K. Tang, & X. Zhang (Eds.), *Bio-Inspired Computing -- Theories and Applications: 10th International Conference, BIC-TA 2015 Hefei, China, September 25-28, 2015, Proceedings* (pp. 315-330). Berlin, Heidelberg: Springer Berlin Heidelberg.

Puchinger, J., & Raidl, G. R. (2005). *Combining metaheuristics and exact algorithms in combinatorial optimization: a survey and classification*. Paper presented at the First International Work-Conference on the Interplay Between Natural and Artificial Computation, Las Palmas, Spain.

Puente, J., Gómez, A., Fernández, I., & Priore, P. (2009). Medical doctor rostering problem in a hospital emergency department by means of genetic

algorithms. *Computers & Industrial Engineering, 56*(4), 1232-1242. doi:http://dx.doi.org/10.1016/j.cie.2008.07.016

Qu, R., & He, F. (2009). A hybrid constraint programming approach for nurse rostering problems. In *Applications and Innovations in Intelligent Systems XVI* (pp. 211-224): Springer.

Reeves, C. R. (1997). Genetic Algorithms for the Operations Researcher. *INFORMS Journal on Computing, 9*(3), 231-250.

Resende, M. C., Ribeiro, C., Glover, F., & Martí, R. (2010). Scatter Search and Path-Relinking: Fundamentals, Advances, and Applications. In M. Gendreau & J.-Y. Potvin (Eds.), *Handbook of Metaheuristics* (Vol. 146, pp. 87-107): Springer US.

Respício, A., Moz, M., & Pato, M. V. (2013). Enhanced genetic algorithms for a bi-objective bus driver rostering problem: a computational study. *International Transactions in Operational Research, 20*(4), 443-470. doi:10.1111/itor.12013

Rodrigues, M. M., de Souza, C. C., & Moura, A. V. (2006). Vehicle and crew scheduling for urban bus lines. *European Journal of Operational Research, 170*(3), 844-862. doi:10.1016/j.ejor.2004.06.035

Romanycia, M. H. J., & Pelletier, F. J. (1985). What is a heuristic? *Computational Intelligence, 1*(1), 47-58. doi:10.1111/j.1467-8640.1985.tb00058.x

Ross, G. T., & Soland, R. (1975). A branch and bound algorithm for the generalized assignment problem. *Mathematical Programming, 8*(1), 91-103. doi:10.1007/bf01580430

Rossi, F., van Beek, P., & Walsh, T. (2006). *Handbook of Constraint Programming*: Elsevier Science.

Ruibin, B., Burke, E. K., Kendall, G., Jingpeng, L., & McCollum, B. (2010). A Hybrid Evolutionary Approach to the Nurse Rostering Problem. *Evolutionary Computation, IEEE Transactions on, 14*(4), 580-590.

Sabar, M., Montreuil, B., & Frayret, J. M. (2009). A multi-agent-based approach for personnel scheduling in assembly centers. *Engineering Applications of Artificial Intelligence, 22*(7), 1080-1088. doi:http://dx.doi.org/10.1016/j.engappai.2009.02.009

Santos, D., de Sousa, A., & Alvelos, F. (2013). A hybrid column generation with GRASP and path relinking for the network load balancing problem. *Computers & Operations Research, 40*(12), 3147-3158. doi:http://dx.doi.org/10.1016/j.cor.2013.05.006

Santos, D., de Sousa, A., Alvelos, F., & Pióro, M. (2013). Optimizing network load balancing: an hybridization approach of metaheuristics with column generation. *Telecommunication Systems, 52*(2), 959-968. doi:10.1007/s11235-011-9604-3

Sargut, F. Z., Altuntaş, C., & Tulazoğlu, D. C. (2017). Multi-objective integrated acyclic crew rostering and vehicle assignment problem in public bus transportation. *OR Spectrum*. doi:10.1007/s00291-017-0485-z

Sellmann, M., Zervoudakis, K., Stamatopoulos, P., & Fahle, T. (2002). Crew Assignment via Constraint Programming: Integrating Column Generation and Heuristic Tree Search. *Annals of Operations Research, 115*(1), 207-225. doi:10.1023/a:1021105422248

Shibghatullah, A. S., Eldabi, T., & Kuljis, J. (2006, 3-6 Dec. 2006). *A Proposed Multiagent Model for Bus Crew Scheduling.* Paper presented at the Simulation Conference, 2006. WSC 06. Proceedings of the Winter.

Smet, P., Bilgin, B., De Causmaecker, P., & Vanden Berghe, G. (2014). Modelling and evaluation issues in nurse rostering. *Annals of Operations Research, 218*(1), 303-326. doi:10.1007/s10479-012-1116-3

Smet, P., Ernst, A. T., & Vanden Berghe, G. (2016). Heuristic decomposition approaches for an integrated task scheduling and personnel rostering problem. *Computers & Operations Research, 76*, 60-72. doi:http://dx.doi.org/10.1016/j.cor.2016.05.016

Smith, R. G. (1980). The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *Computers, IEEE Transactions on, C-29*(12), 1104-1113. doi:10.1109/tc.1980.1675516

Souai, N., & Teghem, J. (2009). Genetic algorithm based approach for the integrated airline crew-pairing and rostering problem. *European Journal of Operational Research, 199*(3), 674-683. doi:http://dx.doi.org/10.1016/j.ejor.2007.10.065

Stølevik, M., Nordlander, T., Riise, A., & Frøyseth, H. (2011). A Hybrid Approach for Solving Real-World Nurse Rostering Problems. In J. Lee (Ed.), *Principles and Practice of Constraint Programming – CP 2011* (Vol. 6876, pp. 85-99): Springer Berlin Heidelberg.

Talbi, E.-G. (2009a). Population-Based Metaheuristics. In *Metaheuristics* (pp. 190-307): John Wiley & Sons, Inc.

Talbi, E.-G. (2009b). Single-Solution Based Metaheuristics. In *Metaheuristics* (pp. 87-189): John Wiley & Sons, Inc.

Talbi, E. G. (2009c). *Metaheuristics: From Design to Implementation*: Wiley.

Topaloglu, S. (2006). A multi-objective programming model for scheduling emergency medicine residents. *Computers & Industrial Engineering, 51*(3), 375-388. doi:http://dx.doi.org/10.1016/j.cie.2006.08.003

Van den Bergh, J., Beliën, J., De Bruecker, P., Demeulemeester, E., & De Boeck, L. (2013). Personnel scheduling: A literature review. *European Journal of Operational Research, 226*(3), 367-385. doi:http://dx.doi.org/10.1016/j.ejor.2012.11.029

van der Veen, E., Hans, E., Post, G., & Veltman, B. (2015). Shift rostering using decomposition: assign weekend shifts first. *Journal of Scheduling, 18*(1), 29-43. doi:10.1007/s10951-014-0385-x

Vasconcelos, J. A., Ramirez, J. A., Takahashi, R. H. C., & Saldanha, R. R. (2001). Improvements in genetic algorithms. *Magnetics, IEEE Transactions on, 37*(5), 3414-3417. doi:10.1109/20.952626

Vohra, R. V. (1987). The Cost of Consecutivity in the (5, 7) Cyclic Staffing Problem. *IIE Transactions, 19*(3), 296-299. doi:10.1080/07408178708975399

Walsh, T. (2006). Symmetry Breaking. In A. Sattar & B.-h. Kang (Eds.), *AI 2006: Advances in Artificial Intelligence* (Vol. 4304, pp. 7-8): Springer Berlin Heidelberg.

Walsh, T. (2012). *Symmetry Breaking Constraints: Recent Results*. Paper presented at the Twenty-Sixth AAAI Conference on Artificial Intelligence. http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/4974

Wang, Z., & Wang, C. (2009, 11-14 Oct. 2009). *Automating nurse self-rostering: A multiagent systems model.* Paper presented at the Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on.

Wilhelm, W. (2001). A Technical Review of Column Generation in Integer Programming. *Optimization and Engineering, 2*(2), 159-200. doi:10.1023/a:1013141227104

Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on, 1*(1), 67-82. doi:10.1109/4235.585893

Wolsey, L. A. (1998). *Integer Programming*: John Wiley and Sons.

Wooldridge, M. (2009). *An Introduction to MultiAgent Systems*: Wiley.

Wooldridge, M., & Jennings, N. R. (1995). Intelligent agents: Theory and practice. *Knowledge Engineering Review, 10*(2), 115-152.

Wren, A. (1996). Scheduling, timetabling and rostering — A special relationship? In E. Burke & P. Ross (Eds.), *Practice and Theory of Automated Timetabling* (Vol. 1153, pp. 46-75): Springer Berlin / Heidelberg.

Xie, L., Kliewer, N., & Suhl, L. (2012). Integrated Driver Rostering Problem in Public Bus Transit. *Procedia - Social and Behavioral Sciences, 54*(0), 656-665. doi:http://dx.doi.org/10.1016/j.sbspro.2012.09.783

Xie, L., Merschformann, M., Kliewer, N., & Suhl, L. (2017). Metaheuristics approach for solving personalized crew rostering problem in public bus transit. *Journal of Heuristics, 23*(5), 321-347. doi:10.1007/s10732-017-9348-7

Xie, L., & Suhl, L. (2015). Cyclic and non-cyclic crew rostering problems in public bus transit. *OR Spectrum, 37*(1), 99-136. doi:10.1007/s00291-014-0364-9

Yunes, T. H., Moura, A. V., & de Souza, C. C. (2005). Hybrid Column Generation Approaches for Urban Transit Crew Management Problems. *Transportation Science, 39*(2), 273-288. doi:10.1287/trsc.1030.0078

Appendix A – Column Generation Detailed Results

The number of CG iterations used in each instance by each configuration is presented in Table 15. The total execution time is presented by Table 16 and the amount of time spent optimizing the RMP, solving the subproblems with the exact solver and solving the subproblems using heuristics are presented in Table 17, Table 18 and Table 19, respectively. Table 20 displays the number of subproblems solved using the exact solver and Table 21 shows the number of times the heuristic was used. Finally, Table 22 presents the total number of generated columns, which is the search space to be explored by the metaheuristics.

Table 15 – Column generation iterations

INSTANCE	SP.A	SP.N	SP.CP	SP.A_D	SP.N_D	SP.CP_D	SP.A_RR	SP.A_RS	SP.A_RR+	SP.A_RS+	SP.CP_RR+	SP.N_RR+
P80_1	910	729	287	3495	1775	3204	1258	1170	457	805	376	463
P80_2	703	684	375	5945	1548	4335	772	697	420	537	375	579
P80_3	669	539	280	2694	1367	2367	443	407	340	347	247	351
P80_4	856	850	274	3896	2405	4212	1223	1208	658	646	440	803
P80_5	798	806	275	2852	1881	2921	864	803	424	558	393	577
P80_6	787	1158	315	3051	1785	2665	574	544	458	431	367	480
P80_7	915	976	279	2979	1675	4008	774	628	606	684	443	763
P80_8	962	730	224	4853	3488	3636	1189	1080	547	681	505	797
P80_9	776	802	291	3453	2254	3973	873	812	732	719	452	718
P80_10	943	793	218	1576	1369	1660	2832	2990	786	684	340	641
P100_1	1170	543	183	1430	1582	1724	728	693	584	545	470	625
P100_2	986	645	213	1423	1512	1723	758	2050	1072	821	1016	964
P100_3	1046	506	199	1712	1879	1925	1154	1306	564	669	474	651
P100_4	928	906	226	1232	1332	1655	750	667	625	799	403	917
P100_5	1009	660	204	1787	1707	1635	1704	1653	521	607	413	584
P100_6	906	778	250	1050	1075	1047	850	957	812	735	674	664
P100_7	1020	652	223	2471	2104	1953	2458	2726	711	804	479	962
P100_8	1095	626	194	2427	2203	2359	928	745	655	633	419	945
P100_9	1165	476	180	2447	1944	2192	1800	2127	720	778	484	786
P100_10	559	611	282	1133	1803	1189	2009	1852	801	682	714	691
C122_1	779	829	293	5773	1601	5352	997	920	512	491	520	592
C122_2	741	728	307	6387	2061	5916	776	857	504	498	473	560
C122_3	877	546	312	7892	2265	6288	1137	1159	612	613	488	673
C224_1	1120	151	133	1476	1395	1286	1328	1281	934	932	507	551
C224_2	886	110	130	1811	1809	1757	1150	1142	867	908	528	452
C224_3	1131	172	135	1462	1269	1431	1363	1160	948	918	327	370
C226_1	1222	246	160	1968	3711	2349	1512	1197	762	763	315	284
C226_2	1185	150	151	2406	2033	2384	1182	1037	807	754	283	301
C226_3	1190	210	151	2231	1850	1818	1298	1220	856	855	297	271
C238_1	826	102	119	1502	1561	1282	1015	988	536	533	278	244
C238_2	680	104	117	1999	1988	2046	997	973	631	594	366	251
C238_3	917	107	119	1104	1111	1116	1023	995	618	597	240	214

Table 16 –Total time

INSTANCE	SP.A	SP.N	SP.CP	SP.A_D	SP.N_D	SP.CP_D	SP.A_RR	SP.A_RS	SP.A_RR+	SP.A_RS+	SP.CP_RR+	SP.N_RR+
P80_1	687	3337	7228	2493	1873	3782	822	776	328	608	3898	1145
P80_2	304	1443	7219	2106	873	3154	199	187	140	191	2207	762
P80_3	518	4548	7211	1730	1631	2716	192	173	233	222	2719	916
P80_4	656	4027	7204	3011	2058	5364	758	838	560	486	3936	2101
P80_5	548	3609	7211	1604	1793	4814	459	441	291	392	4654	1723
P80_6	469	5171	7204	1556	1122	3242	212	195	239	229	2149	1125
P80_7	793	4885	7402	2503	2355	5683	506	412	500	612	3935	2323
P80_8	1279	7216	7335	7203	4504	7203	1503	1176	719	991	6807	4338
P80_9	550	3254	7264	2234	3037	5403	519	420	532	535	4030	1716
P80_10	1306	7203	7366	2240	1955	3731	7207	7213	1201	992	5604	3480
P100_1	2859	7204	7606	3094	3839	5735	2009	1875	1475	1333	7215	6771
P100_2	1530	7208	7211	2229	2505	4350	1522	5263	2460	1603	5668	2658
P100_3	1935	7212	7229	2920	3123	5929	2710	3238	991	1275	6588	5768
P100_4	1057	7208	7232	1362	1812	2462	1135	1000	825	1300	4910	3791
P100_5	1826	7214	7205	3340	2825	4850	4184	3945	923	1076	6622	4951
P100_6	868	4604	7201	742	909	750	1060	1272	1110	890	1755	1095
P100_7	1745	7206	7214	4597	3228	3773	6290	7208	1282	1552	7238	5648
P100_8	2299	7209	7235	5311	5953	7223	2282	1466	1328	1296	3558	6605
P100_9	2746	7203	7234	6045	5500	7203	6470	7205	1800	1989	7243	7209
P100_10	354	2533	7212	680	884	1066	1625	1457	628	474	1605	702
C122_1	548	4526	7354	3465	1859	7234	484	446	363	327	7078	1756
C122_2	594	4950	7227	3588	2922	7218	354	393	381	362	7209	1851
C122_3	833	7215	7352	4235	5705	7279	656	670	542	545	7222	4744
C224_1	7204	7231	7233	7204	7202	7213	7220	7205	7209	7202	7236	7217
C224_2	7240	7307	7205	7210	7203	7208	7210	7220	6571	7203	7263	7265
C224_3	7207	7257	7207	7207	7224	7205	7203	5869	7202	6911	7257	7245
C226_1	7206	7328	7230	7205	7202	7241	5365	4089	4011	4069	7214	7293
C226_2	7203	7268	7246	7204	7231	7209	4698	3634	4101	4018	7245	7219
C226_3	7211	7225	7238	7204	7204	7205	5867	4926	5542	5609	7252	7317
C238_1	7214	7269	7206	7220	7214	7213	7208	7214	7217	7201	7249	7228
C238_2	7272	7264	7202	7202	7210	7209	7209	7206	7215	7212	7243	7261
C238_3	7208	7255	7262	7209	7214	7207	7203	7206	7214	7204	7271	7316

Table 17 – RMP time

INSTANCE	SP.A	SP.N	SP.CP	SP.A_D	SP.N_D	SP.CP_D	SP.A_RR	SP.A_RS	SP.A_RR+	SP.A_RS+	SP.CP_RR+	SP.N_RR+
P80_1	454	550	87	2426	692	2139	803	755	257	477	202	265
P80_2	161	163	76	2045	297	1542	179	161	102	141	95	155
P80_3	285	375	101	1675	561	1379	173	155	179	170	104	176
P80_4	422	591	68	2934	1195	3250	736	811	476	405	274	547
P80_5	332	463	86	1535	697	1760	429	404	215	283	208	353
P80_6	276	777	96	1490	640	1345	186	166	168	161	136	208
P80_7	558	815	69	2447	923	3546	492	399	442	524	326	610
P80_8	899	882	76	7076	3159	5467	1447	1100	636	850	619	1108
P80_9	352	436	73	2185	986	2838	500	388	439	429	258	468
P80_10	978	1049	61	2200	1477	2297	7184	7189	1098	894	354	810
P100_1	2292	476	68	3030	2888	3396	1998	1857	1302	1201	1092	1464
P100_2	1137	994	55	2183	1937	2731	1513	5239	2347	1489	2171	2354
P100_3	1404	354	59	2862	2580	3766	2688	3211	822	1116	677	1114
P100_4	739	1246	39	1329	1133	2004	1123	985	810	1287	399	1323
P100_5	1361	788	49	3282	2434	2882	4155	3912	768	933	601	916
P100_6	589	712	37	723	623	702	1053	1265	1098	873	817	769
P100_7	1315	732	43	4530	2849	3504	6246	7186	1126	1385	646	1911
P100_8	1745	816	52	5204	3234	5152	2262	1451	1233	1220	651	2197
P100_9	2136	499	41	5915	3405	5362	6441	7184	1602	1797	877	1991
P100_10	201	360	57	661	805	721	1608	1442	616	457	478	553
C122_1	319	447	87	3348	645	3104	440	405	261	237	311	330
C122_2	335	467	116	3465	797	3273	297	327	275	254	243	294
C122_3	388	248	117	4017	950	3036	514	516	338	343	265	362
C224_1	5909	52	20	7088	6573	7055	7199	7188	6605	6598	2330	2530
C224_2	2594	35	25	7057	6921	7055	7087	7162	5905	6547	2494	2150
C224_3	6039	47	17	7091	5370	7090	7000	5636	6395	6127	1327	1813
C226_1	3644	156	171	7097	6872	6717	4864	3634	2976	2851	783	697
C226_2	3613	40	130	7064	5539	6878	4070	3162	3050	2910	804	829
C226_3	3664	109	121	7050	6435	6995	5102	4286	4051	4082	906	809
C238_1	6588	10	63	7032	6892	7043	7070	7029	6509	6497	2043	1603
C238_2	6742	17	179	6949	6743	6941	7115	7064	6680	6585	3007	1831
C238_3	6176	46	117	7061	6991	7058	7079	7033	6649	6622	1824	1421

Table 18 – Exact subproblem solver time

INSTANCE	SP.A	SP.N	SP.CP	SP.A_D	SP.N_D	SP.CP_D	SP.A_RR	SP.A_RS	SP.A_RR+	SP.A_RS+	SP.CP_RR+	SP.N_RR+
P80_1	228	2764	7132	25	1130	1608	11	13	67	127	3695	844
P80_2	141	1268	7137	27	554	1587	17	24	36	48	2111	580
P80_3	228	4161	7104	22	1035	1307	16	16	52	49	2614	709
P80_4	230	3410	7134	33	800	2065	14	19	79	77	3660	1501
P80_5	212	3119	7118	36	1030	3006	25	33	74	106	4444	1327
P80_6	190	4351	7104	41	446	1873	23	26	69	66	2011	872
P80_7	231	4044	7138	17	1388	2091	10	8	54	85	3607	1617
P80_8	372	6310	7147	47	1209	1673	45	67	78	137	6185	3116
P80_9	195	2793	7135	15	1990	2512	14	28	89	102	3770	1194
P80_10	321	6122	7223	3	406	1394	4	4	97	92	5247	2603
P100_1	556	6708	7524	2	842	2256	2	2	166	124	6121	5156
P100_2	383	6188	7156	2	479	1573	1	4	103	107	3488	112
P100_3	518	6837	7169	6	411	2102	5	5	163	151	5907	4443
P100_4	311	5927	7192	3	622	418	2	2	5	6	4508	2370
P100_5	454	6397	7156	3	277	1920	4	4	150	137	6016	3838
P100_6	272	3866	7163	0	253	29	1	1	7	12	934	266
P100_7	421	6446	7170	2	206	211	5	3	149	159	6590	3603
P100_8	543	6368	7182	36	2568	2010	8	8	84	68	2903	4178
P100_9	590	6687	7192	40	1950	1771	3	5	187	179	6364	5190
P100_10	150	2158	7154	2	28	330	2	3	6	13	1124	99
C122_1	225	4054	7127	60	1171	4077	38	37	98	87	6765	1395
C122_2	255	4466	7106	66	2077	3888	52	61	104	106	6964	1526
C122_3	441	6955	7089	147	4691	4131	136	148	202	199	6956	4321
C224_1	1288	7150	7212	1	454	54	5	4	597	598	4901	4658
C224_2	4641	7236	7179	0	0	0	111	46	633	648	4764	5086
C224_3	1162	7174	7190	0	1700	0	190	197	802	763	5928	5413
C226_1	3556	7149	7058	1	47	398	465	436	1023	1204	6429	6581
C226_2	3584	7203	7115	3	1401	189	605	445	1037	1094	6439	6378
C226_3	3541	7107	7116	15	503	95	738	614	1474	1510	6344	6497
C238_1	618	7253	7142	0	0	0	122	172	703	700	5202	5608
C238_2	524	7241	7021	0	0	0	80	129	529	622	4232	5412
C238_3	1023	7203	7144	0	0	0	110	161	560	577	5444	5881

Table 19 – Heuristic subproblem solver time

INSTANCE	SP.A	SP.N	SP.CP	SP.A_D	SP.N_D	SP.CP_D	SP.A_RR	SP.A_RS	SP.A_RR+	SP.A_RS+	SP.CP_RR+	SP.N_RR+
P80_1	0	0	0	28	14	26	2	1	0	0	0	0
P80_2	0	0	0	25	6	18	1	1	0	0	0	0
P80_3	0	0	0	25	12	23	1	0	0	0	0	0
P80_4	0	0	0	32	20	35	2	1	0	0	0	1
P80_5	0	0	0	23	14	23	1	1	0	0	0	0
P80_6	0	0	0	19	12	17	1	0	0	0	0	0
P80_7	0	0	0	27	13	36	1	1	1	0	0	1
P80_8	0	0	0	72	56	56	2	2	1	1	1	1
P80_9	0	0	0	26	14	30	1	1	0	0	0	1
P80_10	0	0	0	28	26	30	6	6	1	1	0	1
P100_1	0	0	0	45	52	53	2	2	1	1	1	1
P100_2	0	0	0	30	33	36	2	4	2	1	2	3
P100_3	0	0	0	39	47	42	3	3	1	1	1	1
P100_4	0	0	0	21	23	28	1	1	1	1	0	2
P100_5	0	0	0	41	41	37	4	4	0	1	1	1
P100_6	0	0	0	14	13	14	1	1	1	1	1	1
P100_7	0	0	0	52	47	41	5	6	1	1	1	2
P100_8	0	0	0	55	52	55	2	2	1	1	1	2
P100_9	0	0	0	69	59	65	5	5	1	1	1	2
P100_10	0	0	0	11	18	11	2	2	1	1	1	1
C122_1	0	0	0	41	11	37	1	1	0	0	0	0
C122_2	0	0	0	43	12	41	1	1	0	0	0	0
C122_3	0	0	0	55	14	46	1	1	0	0	0	0
C224_1	0	0	0	109	108	98	5	5	2	2	2	3
C224_2	0	0	0	146	156	146	5	5	2	3	2	3
C224_3	0	0	0	109	98	109	6	4	1	1	1	2
C226_1	0	0	0	93	186	114	5	3	1	0	1	1
C226_2	0	0	0	129	117	133	3	3	1	1	1	1
C226_3	0	0	0	131	114	110	4	4	1	1	1	1
C238_1	0	0	0	175	191	159	6	6	1	1	1	2
C238_2	0	0	0	236	250	251	6	6	2	1	2	2
C238_3	0	0	0	130	139	136	6	6	1	1	1	2

Table 20 – Number of subproblems solved with exact solver

INSTANCE	SP.A	SP.N	SP.CP	SP.A_D	SP.N_D	SP.CP_D	SP.A_RR	SP.A_RS	SP.A_RR+	SP.A_RS+	SP.CP_RR+	SP.N_RR+
P80_1	22750	18225	7150	2375	8050	1600	1025	1250	6450	12900	3675	5800
P80_2	14060	13680	7480	2640	6180	1580	1620	2400	3540	4800	2100	6520
P80_3	17394	14014	7254	1638	5850	1300	1248	1196	3744	3666	2600	3744
P80_4	22256	22100	7098	3042	5668	2054	1326	1794	7566	7618	3640	10608
P80_5	20748	20956	7124	3146	7358	2990	2158	2886	6786	10010	4420	9308
P80_6	18101	26634	7222	3657	4048	1863	2001	2185	6601	6164	2001	7222
P80_7	23790	25376	7228	1664	9334	2080	884	780	5382	8554	3588	11076
P80_8	30784	23328	7136	3712	5472	1664	3456	5568	6112	10848	6144	12800
P80_9	19400	20050	7250	1425	15350	2500	1325	2750	9025	10325	3750	9050
P80_10	31119	26136	7161	330	1749	1386	374	429	9768	9306	5214	10626
P100_1	44460	20596	6916	152	2394	2242	152	152	13756	10298	6080	13946
P100_2	33524	21896	7208	136	1836	1564	68	306	8840	9316	3468	442
P100_3	37656	18180	7128	468	1440	2088	380	396	12852	11880	5868	13464
P100_4	29696	28960	7200	320	2976	416	160	192	512	576	4480	10880
P100_5	36324	23724	7308	252	972	1908	370	360	12672	11448	5976	12996
P100_6	26274	22562	7221	29	1479	29	116	116	696	1218	928	1595
P100_7	35700	22785	7770	175	770	210	468	280	13300	14105	6545	13265
P100_8	40515	23125	7141	2849	8103	1998	592	629	6549	5661	2886	11877
P100_9	46600	19000	7160	3080	5520	1760	246	400	15200	14600	6320	13960
P100_10	15652	17108	7868	252	224	336	224	280	672	1428	1120	784
C122_1	19475	20725	7300	4675	7175	4050	2950	2825	7775	6925	6725	8400
C122_2	17043	16744	7038	4807	11339	3864	3703	4186	7038	7429	6923	8717
C122_3	21048	13080	7464	8136	12624	4104	7440	8112	10248	10104	6912	10872
C224_1	60426	8100	7128	54	594	54	270	216	29592	29538	4860	5130
C224_2	48675	5995	7095	0	0	0	4070	1870	20350	19085	4730	3465
C224_3	59890	9063	7102	0	2120	0	9646	10123	38743	38001	5883	5883
C226_1	53724	10780	6996	44	88	396	14432	13508	25872	28116	6380	4136
C226_2	55648	7003	7050	141	1833	188	18377	13912	28247	28200	6392	4700
C226_3	55883	9823	7050	799	705	94	20586	17014	33276	33417	6298	3995
C238_1	49500	6060	7080	0	0	0	4800	6900	25140	25020	5160	3120
C238_2	40740	6180	6960	0	0	0	3600	5640	20580	24120	4200	3060
C238_3	54960	6360	7080	0	0	0	4860	7140	23880	24780	5400	3360

Table 21 – Number of heuristic solver runs[1]

INSTANCE	SP.A	SP.N	SP.CP	SP.A_D	SP.N_D	SP.CP_D	SP.A_RR	SP.A_RS	SP.A_RR+	SP.A_RS+	SP.CP_RR+	SP.N_RR+
P80_1	0	0	0	85000	36325	78500	1217	1120	199	289	229	231
P80_2	0	0	0	116260	24780	85120	691	577	243	297	270	253
P80_3	0	0	0	68406	29692	60242	395	361	196	206	147	207
P80_4	0	0	0	98254	56862	107458	1172	1139	367	353	300	395
P80_5	0	0	0	71006	41548	72956	781	692	163	173	223	219
P80_6	0	0	0	66516	37007	59432	487	449	171	163	280	166
P80_7	0	0	0	75790	34216	102128	740	598	399	355	305	337
P80_8	0	0	0	151552	106144	114656	1081	906	356	342	313	397
P80_9	0	0	0	84900	41000	96825	820	702	371	306	302	356
P80_10	0	0	0	51678	43428	53394	2820	2976	490	402	182	319
P100_1	0	0	0	54188	57722	63270	724	689	222	274	309	258
P100_2	0	0	0	48246	49572	57018	756	2041	812	547	914	951
P100_3	0	0	0	61164	66204	67212	1144	1295	207	339	311	277
P100_4	0	0	0	39104	39648	52544	745	661	609	781	263	577
P100_5	0	0	0	64080	60480	56952	1694	1643	169	289	247	223
P100_6	0	0	0	30421	29696	30334	846	953	788	693	642	609
P100_7	0	0	0	86310	72870	68145	2445	2717	331	401	291	583
P100_8	0	0	0	86950	73408	85248	912	728	478	480	341	624
P100_9	0	0	0	94800	72240	85880	1794	2116	340	413	325	436
P100_10	0	0	0	31472	50260	32956	2001	1842	777	631	674	663
C122_1	0	0	0	139650	32850	129725	879	807	201	214	251	256
C122_2	0	0	0	142094	36064	132181	615	675	198	175	171	181
C122_3	0	0	0	181272	41736	146784	827	821	185	192	199	220
C224_1	0	0	0	79596	74682	69336	1322	1276	385	384	416	455
C224_2	0	0	0	99550	99440	96580	1075	1107	497	560	441	388
C224_3	0	0	0	77433	65084	75790	1180	969	216	201	215	258
C226_1	0	0	0	86504	163152	102916	1184	890	174	124	169	189
C226_2	0	0	0	112894	93671	111813	791	741	206	154	146	200
C226_3	0	0	0	104011	86198	85305	860	858	148	144	162	185
C238_1	0	0	0	90060	93600	76860	934	872	116	115	191	191
C238_2	0	0	0	119880	119220	122700	936	878	287	191	295	199
C238_3	0	0	0	66180	66600	66900	941	875	219	183	149	157

[1] The Roster Heuristic solves multiples subproblems in each iteration. The number of subproblems solved is greater than the number of runs. In the limit, a single run solves all the subproblems together.

Table 22 – Number of subproblem solutions (columns generated)

INSTANCE	SP.A	SP.N	SP.CP	SP.A_D	SP.N_D	SP.CP_D	SP.A_RR	SP.A_RS	SP.A_RR+	SP.A_RS+	SP.CP_RR+	SP.N_RR+
P80_1	21289	17286	7225	62718	29430	57593	30776	28474	10449	16691	8901	11160
P80_2	13825	13640	7540	71196	20053	56085	14799	13346	7802	9656	6859	10398
P80_3	17444	14000	7332	45336	24010	40239	10984	9917	8307	8430	5897	8647
P80_4	21188	20941	7176	59360	28852	64938	30568	29998	15418	15406	10917	18357
P80_5	20470	19999	7202	54810	33902	55586	20959	19465	10314	13000	9331	13846
P80_6	17454	26285	7291	49520	23687	47402	12434	11849	9030	8854	7785	10115
P80_7	22332	23009	7306	58467	27031	67522	19347	15680	14211	15010	10996	16617
P80_8	30723	23419	7232	97372	48558	80763	35040	30592	16614	19839	15455	23158
P80_9	19371	19064	7325	61625	31053	67121	20273	18855	15707	15416	10660	15381
P80_10	30704	25393	7260	49981	38480	49856	93235	96028	24882	21482	10684	20366
P100_1	44312	20706	7030	52366	52109	56024	26979	25674	21191	19963	16515	23117
P100_2	33241	21998	7310	47152	43191	52126	25291	68855	35426	27098	33799	32190
P100_3	37146	18288	7236	57836	46440	61221	41069	45893	19402	23176	16340	22967
P100_4	28744	27466	7296	38639	35028	50031	23074	20228	18922	24086	11993	26445
P100_5	36023	23832	7416	62552	45143	52607	60579	58692	17832	20842	14349	20369
P100_6	26135	22186	7308	30345	27732	30316	23681	26513	22374	19841	18474	18093
P100_7	35636	22890	7875	84546	56221	67067	86391	94509	24034	27303	16204	32534
P100_8	40197	23236	7252	72779	53583	73539	32867	26297	23141	22303	14394	33135
P100_9	45494	19120	7280	85181	58777	78767	71116	82477	27327	29878	18607	30238
P100_10	15636	17058	7952	31002	31339	32318	48637	45197	20066	16980	17662	17252
C122_1	18965	19518	7374	70015	24280	66133	23619	21522	11673	11344	12176	13715
C122_2	16986	16443	7106	72501	25406	69829	16897	19060	10898	10788	10351	11945
C122_3	20849	13149	7535	80031	30479	67788	26489	26529	13950	13943	11100	14452
C224_1	60534	8262	7290	79557	71833	69496	69337	67033	49200	49239	26521	29210
C224_2	48785	6160	7260	99646	99368	96731	61975	61666	45143	48517	28036	24460
C224_3	59996	9222	7261	76125	60404	74325	70827	60100	43986	42685	16491	19155
C226_1	53812	10912	7128	86119	55916	98980	66150	52251	33192	33041	13677	12533
C226_2	54457	7144	7191	112805	93338	111469	54177	47600	34833	33944	12467	13708
C226_3	55977	9892	7191	100271	82371	85092	60610	56808	39436	39289	13050	12711
C238_1	49620	6240	7260	90089	93187	76988	60866	59236	32119	31936	16674	14680
C238_2	40860	6353	7140	119961	119064	122818	59795	58320	37837	35580	21987	15105
C238_3	55080	6540	7260	66275	65974	67074	61225	59547	36887	35667	14240	12857

Figure 45 includes the charts for the instances p80, the charts for the instances p100 are in Figure 46 and Figure 47 displays the charts for the group c.

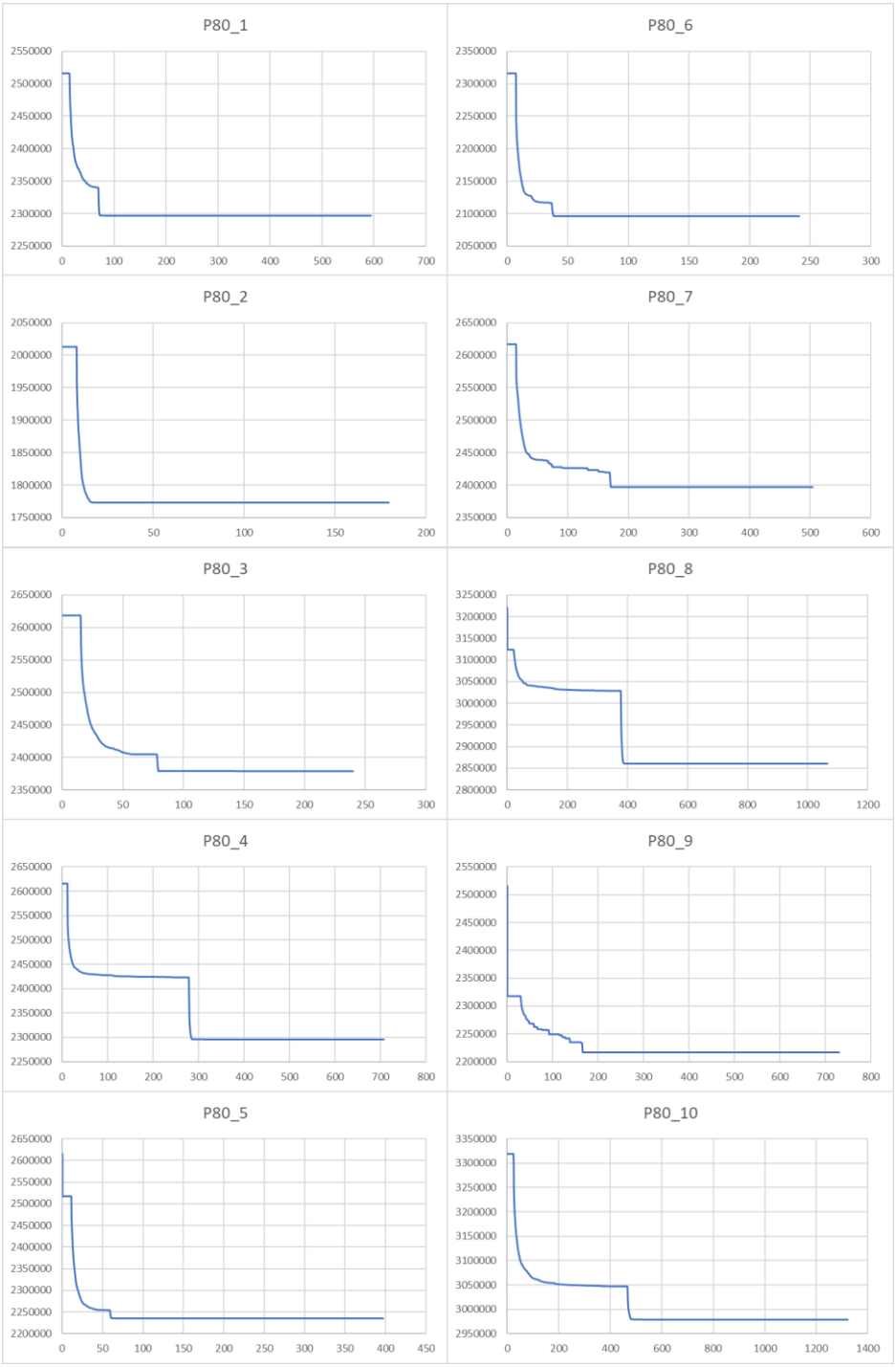

Figure 45 – CG solution value evolution - Instances P80

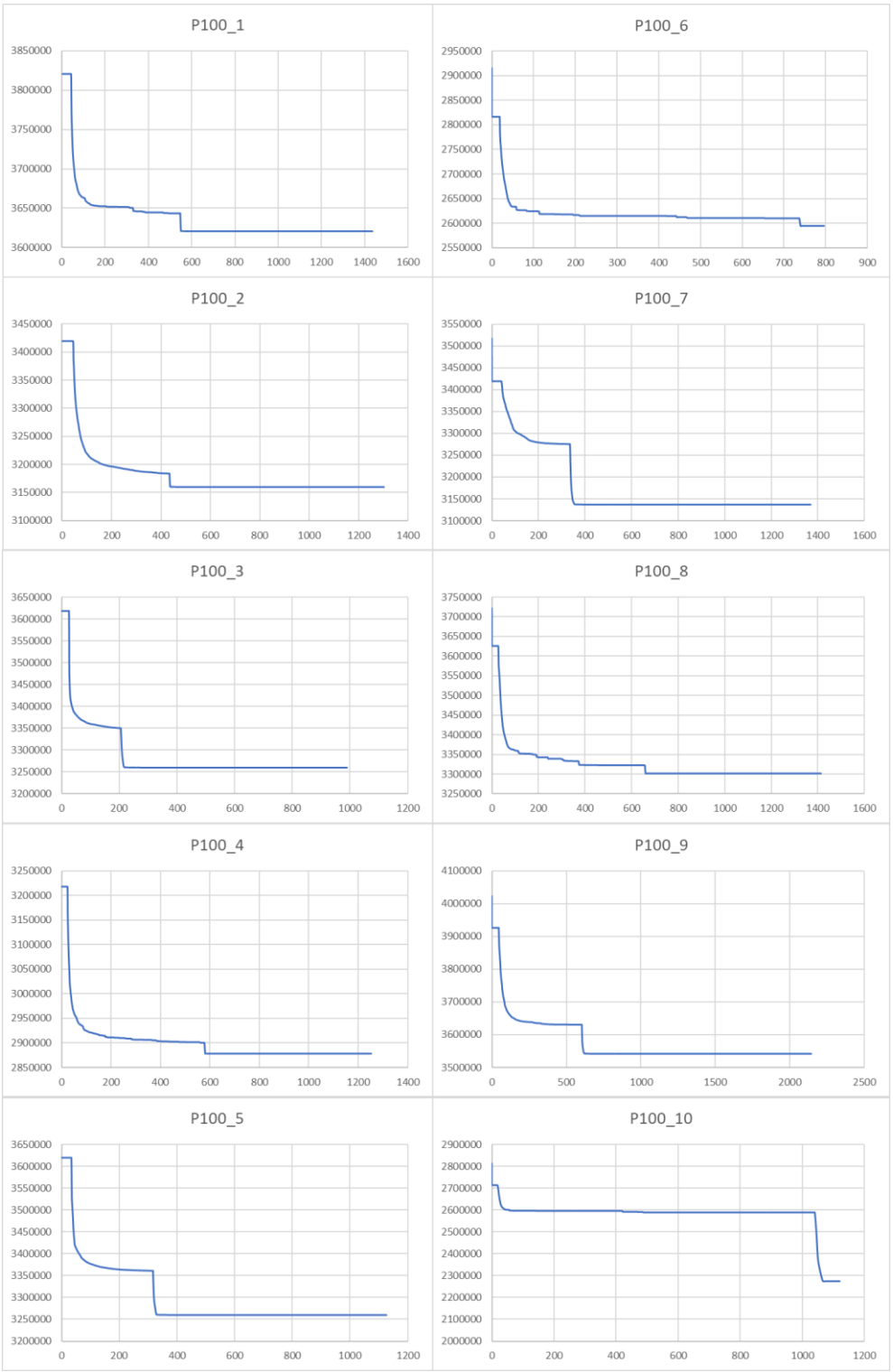

Figure 46 – CG solution value evolution - Instances P100

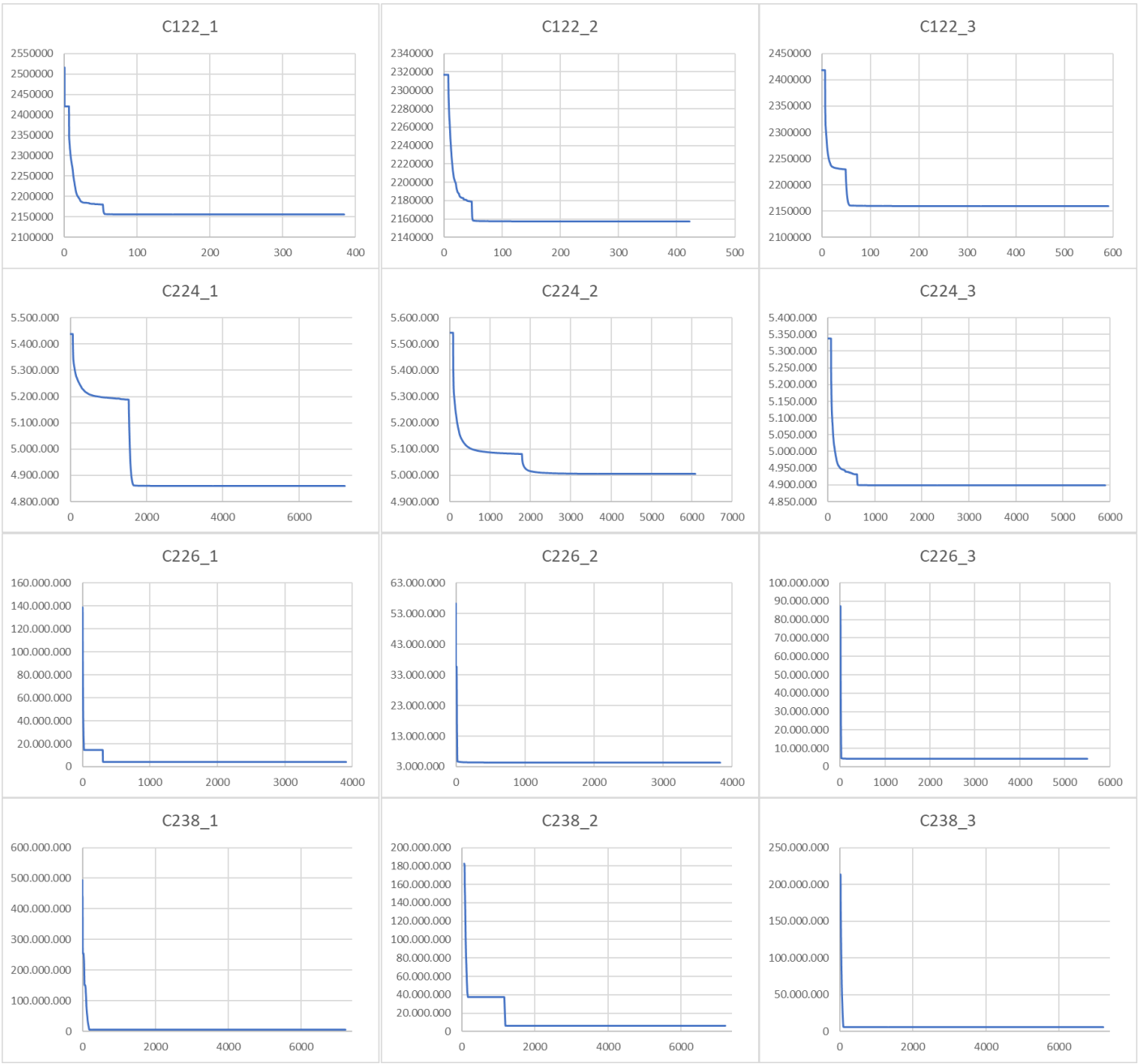

Figure 47 – CG solution value evolution - Instances C

Appendix B – Metaheuristic Search Results

Table 23 – Best infeasibility value

Instance	EA	MIPsearch	SA	VNS
p80_1	400	100360	400	55480
p80_2	37120	55480	400	400
p80_3	60580	83020	57520	63640
p80_4	49560	87300	44460	50580
p80_5	45480	72000	600	44460
p80_6	52420	67720	400	400
p80_7	63640	86080	55480	62620
p80_8	65880	111780	600	600
p80_9	50580	600	42420	49560
p80_10	72000	121980	66900	73020
p100_1	107500	400	102400	90160
p100_2	84040	110560	79960	84040
p100_3	79140	134220	77100	83220
p100_4	69960	94440	61800	68940
p100_5	80160	146460	400	86280
p100_6	61800	77100	55680	61800
p100_7	73020	179100	70980	78120
p100_8	59960	159920	64040	56900
p100_9	82400	124220	75260	83420
p100_10	34660	50980	30580	35680
c122_1	42420	35280	600	43440
c122_2	56300	67520	33860	57320
c122_3	46300	63640	45270	48340
c224_1	120340	281500	124420	123400
c224_2	128700	281700	129720	129720
c224_3	129320	288440	135440	132380
c226_1	118720	248260	114640	124840
c226_2	114020	243560	108920	118100
c226_3	121980	261720	120960	126060
c238_1	208080	355980	199920	213180
c238_2	197060	355160	198080	207260
c238_3	190120	359440	182980	196240

Table 23 contains the best infeasibility value found by each metaheuristic in the 30 test runs.

Table 24 – Average infeasibility value

Instance	EA	MIPsearch	SA	VNS
p80_1	60920	100360	40792	58948
p80_2	44192	55480	37086	40996
p80_3	67652	83020	61124	67006
p80_4	54830	87300	48506	54592
p80_5	50512	72000	42794	49968
p80_6	57384	67720	400	9138
p80_7	67176	86080	60410	65782
p80_8	71592	111780	59692	60406
p80_9	54150	600	47112	52994
p80_10	77542	121980	70368	76148
p100_1	113824	400	107296	113450
p100_2	89072	110560	83938	87610
p100_3	88728	134220	80466	88558
p100_4	77304	94440	67784	73870
p100_5	89170	146460	78249	90258
p100_6	65880	77100	58434	64758
p100_7	82234	179100	74618	81826
p100_8	83760	159920	64040	64210
p100_9	88826	124220	80156	88146
p100_10	42446	50980	33674	39624
c122_1	47282	35280	39088	45684
c122_2	61740	67520	54464	59224
c122_3	53983	63640	47319	51467
c224_1	126800	281500	127718	130846
c224_2	136894	281700	135500	139308
c224_3	137548	288440	140642	141934
c226_1	127016	248260	119638	128750
c226_2	121772	243560	113476	122860
c226_3	129120	261720	125176	132724
c238_1	218212	355980	203524	219878
c238_2	208280	355160	203554	213040
c238_3	197668	359440	189168	200048

Table 24 contains the average infeasibility value from the 30 test runs with each instance obtained by the four metaheuristics.

Table 25 – Search time

Instance	EA	MIPsearch	SA	VNS
p80_1	33,0	1800,2	119,7	87,9
p80_2	15,1	1800,1	65,7	73,4
p80_3	29,9	1800,1	111,2	104,9
p80_4	45,5	1800,1	108,2	194,9
p80_5	33,8	1798,7	103,6	141,9
p80_6	22,7	1800,1	100,3	70,8
p80_7	42,0	1800,1	114,0	192,3
p80_8	74,3	1800,1	170,1	363,1
p80_9	43,3	1800,1	104,6	192,3
p80_10	120,5	1800,1	186,2	425,5
p100_1	138,9	1800,1	233,8	611,6
p100_2	127,7	1800,1	212,1	681,8
p100_3	82,8	1800,2	219,1	483,8
p100_4	57,9	1800,1	170,6	331,8
p100_5	97,3	1800,1	215,7	444,6
p100_6	56,5	1800,1	159,8	371,8
p100_7	119,2	1800,2	207,5	541,4
p100_8	121,9	1800,1	261,6	437,0
p100_9	170,3	1800,1	267,1	676,7
p100_10	52,8	1800,1	128,5	265,6
c122_1	30,4	1800,1	100,8	135,9
c122_2	23,4	1800,1	89,2	80,7
c122_3	29,4	1798,8	92,8	159,8
c224_1	484,3	1800,3	535,5	609,1
c224_2	436,4	1800,3	540,8	609,1
c224_3	444,3	1800,3	488,1	607,0
c226_1	287,6	1800,1	335,1	603,3
c226_2	314,4	1800,2	383,1	604,3
c226_3	383,5	1800,2	387,4	605,7
c238_1	459,5	1801,0	614,2	606,1
c238_2	498,8	1800,4	620,5	609,4
c238_3	438,0	1800,4	617,1	609,1

Table 25 shows the average time used by each metaheuristic in the 30 test runs.

To better understand the distribution of the results obtained by the basic EA in the 30 test runs, we created box plots with the infeasibility values dispersion. Figure 48 includes the boxes for the group of instances p80, Figure 49 the ones for the group p100 and Figure 50 the ones for the group c.

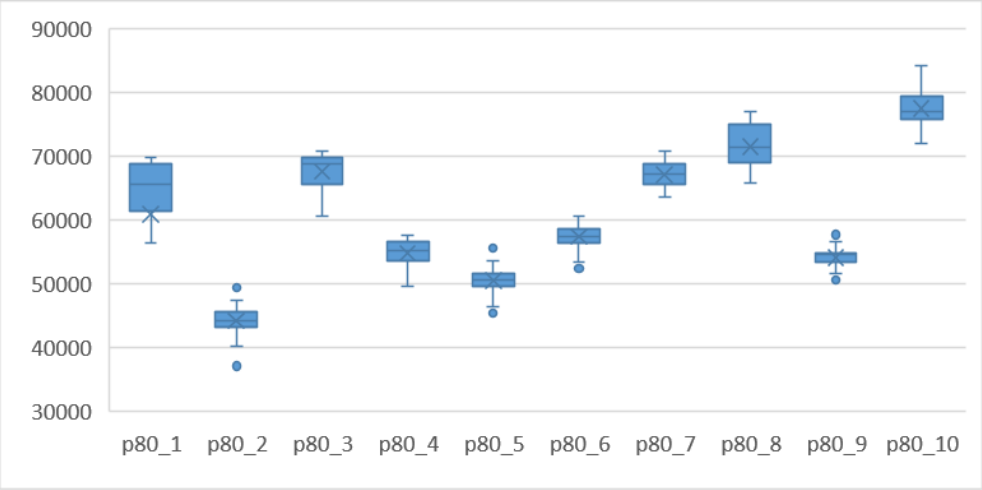

Figure 48 – Infeasibility value dispersion (Instances P80)

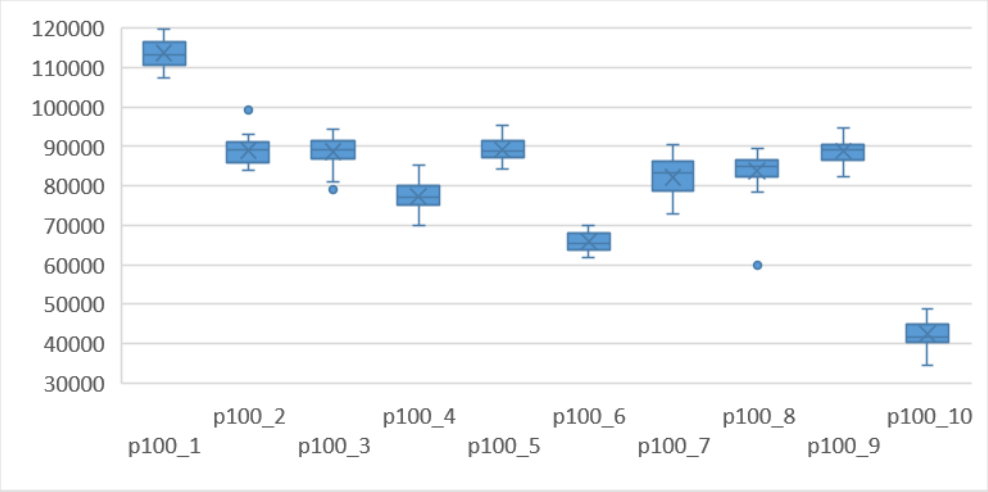

Figure 49 – Infeasibility value dispersion (Instances P100)

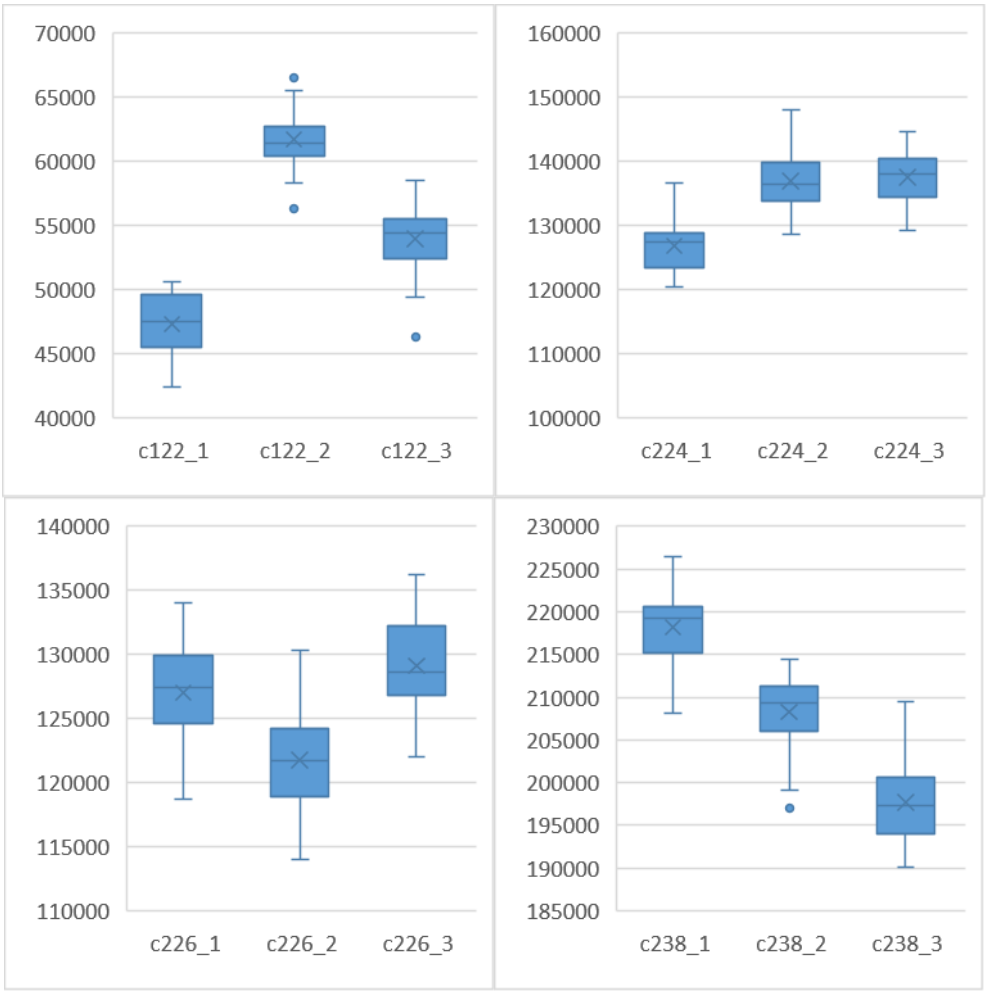

Figure 50 – Infeasibility value dispersion (Instances C)

Appendix C – Evolutionary Algorithm with Repair Results

Table 26 – EA with repair: feasibility and infeasibility results

	Best Solution Value		Average Solution Value		Min Infeasibility Value		Avg Infeasibility Value		Avg Time	
Instance	**C1**	**C2**	**C1**	**C2**	**C1**	**C2**	**C1**	**C2**	**C1**	**C2**
p80_1	2316543	2316564	2330017	2316645	0	0	26,7	0,0	27,8	13,9
p80_2	1812692	1812692	1812738	1812739	0	0	0,0	0,0	18,3	11,7
p80_3	2419156	2419149	2422591	2425963	0	0	6,7	13,3	40,4	15,8
p80_4	2316440	2316454	2403635	2370114	0	0	173,3	106,7	42,4	20,2
p80_5	2315206	2315149	2315256	2315260	0	0	0,0	0,0	110,7	68,6
p80_6	2115818	2115836	2119243	2115913	0	0	6,7	0,0	21,3	10,1
p80_7	2417442	2417450	2487755	2501093	0	0	140,0	166,7	45,4	20,5
p80_8	2920965	2920980	2944538	2924451	0	0	46,7	6,7	150,4	64,0
p80_9	2216393	2216368	2387016	2323384	0	0	340,0	213,3	70,6	17,6
p80_10	3019911	3019911	3026759	3020124	0	0	13,3	0,0	437,1	159,4
p100_1	3620855	3620881	3620953	3621007	0	0	0,0	0,0	1020,0	364,2
p100_2	3219732	3219763	3219823	3219839	0	0	0,0	0,0	741,0	409,3
p100_3	3319154	3319177	3319235	3319281	0	0	0,0	0,0	1135,1	458,8
p100_4	2918149	2918118	2918208	2918232	0	0	0,0	0,0	406,8	164,4
p100_5	3319564	3319580	3319635	3319666	0	0	0,0	0,0	867,5	328,2
p100_6	2615214	2615225	2615271	2615289	0	0	0,0	0,0	375,2	140,3
p100_7	3216309	3216341	3216361	3216393	0	0	0,0	0,0	1367,5	462,1
p100_8	3322291	3322355	3322450	3322470	0	0	0,0	0,0	705,6	291,0
p100_9	3622050	3622086	3622221	3622252	0	0	0,0	0,0	1132,2	464,2
p100_10	2313365	2313377	2330110	2380212	0	0	33,3	133,3	189,4	66,9
c122_1	2216755	2216899	2217033	2217073	0	0	0,0	0,0	54,9	26,5
c122_2	2217951	2217957	2218163	2218147	0	0	0,0	0,0	37,4	18,7
c122_3	2219435	2219424	2229822	2226434	0	0	20,0	13,3	42,3	15,3
c224_1	4940156	4940172	4940385	4940477	0	0	0,0	0,0	1231,9	94,3
c224_2	5244932	5244975	5245134	5245137	600	600	600,0	600,0	1266,7	120,5
c224_3	4939380	4939414	4946252	4942965	0	0	13,3	6,7	1009,3	75,8
c226_1	4239915	4339894	4357092	4340291	0	200	233,3	200,0	262,4	33,2
c226_2	4440727	4440971	4468057	4468176	200	200	253,3	253,3	433,8	39,9
c226_3	4440854	4442204	4532422	4532656	0	0	186,7	180,0	334,9	53,1
c238_1	6052970	6053113	6053308	6053492	0	0	0,0	0,0	1774,9	206,8
c238_2	5952222	6051728	6041940	6051908	0	200	180,0	200,0	1333,8	123,2
c238_3	5951162	5951287	5951400	5951489	200	200	200,0	200,0	1289,3	99,2
All			3404838	3402143			77,3	71,7	561,8	139,3

Table 26 displays the feasibility and infeasibility results (minimum and average) for both configurations (C1 and C2).

Table 27 – Population size, number of duties and drivers in configurations

Instance	# Duties	Pop. Size C1	Pop. Size C2	# Drivers
p80_1	456	402	252	25
p80_2	352	322	202	20
p80_3	472	418	262	26
p80_4	456	418	262	26
p80_5	444	418	262	26
p80_6	416	370	232	23
p80_7	476	418	262	26
p80_8	568	514	322	32
p80_9	440	402	252	25
p80_10	592	530	332	33
p100_1	720	610	382	38
p100_2	628	546	342	34
p100_3	648	578	362	36
p100_4	572	514	322	32
p100_5	648	578	362	36
p100_6	516	466	292	29
p100_7	624	562	352	35
p100_8	656	594	372	37
p100_9	704	642	402	40
p100_10	452	450	282	28
c122-1	428	402	252	25
c122-2	428	370	232	23
c122-3	428	386	242	24
c224-1	964	866	218	54
c224-2	980	882	222	55
c224-3	972	850	214	53
c226-1	824	706	178	44
c226-2	852	754	190	47
c226-3	872	754	190	47
c238-1	1188	962	240	60
c238-2	1180	962	240	60
c238-3	1160	962	240	60

Table 27 shows the information about the instances, concretely the number of duties and the number of drivers considered, and the population size of each configuration.

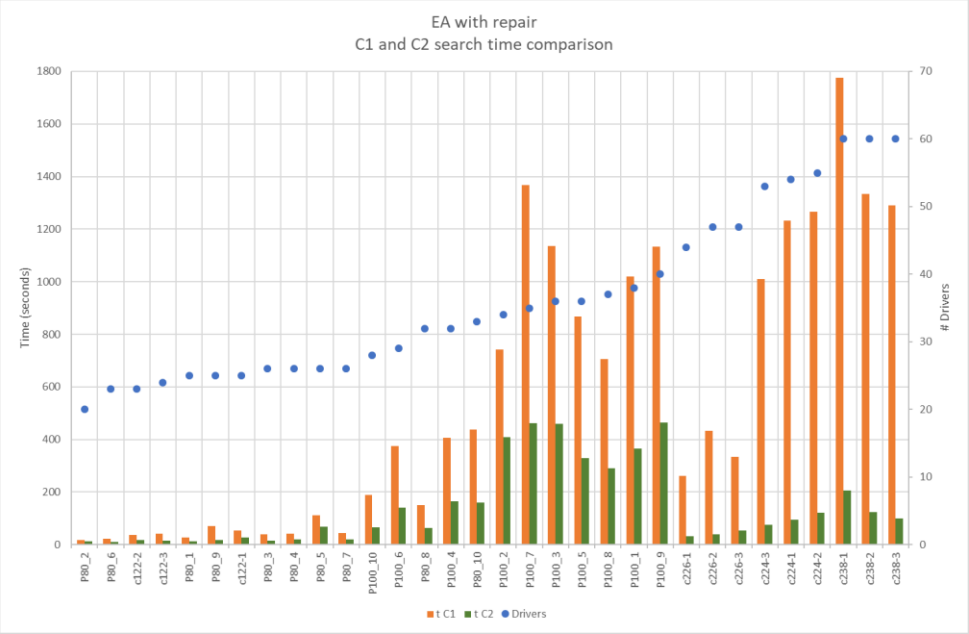

Figure 51 – EA with repair operator: C1 and C2 search time vs number of drivers

Figure 51 illustrates the evolution of the search time of the configurations C1 and C2 with the instances ordered by the number of drivers used.